SHORTEST PATH PROBLEM WITH RE-ROUTING EN-ROUTE

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Banu Karakaya

August, 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Osman Alp (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Bahar Yetiş Kara

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Dr. Banu Yüksel Özkaya

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of the Institute

ABSTRACT

SHORTEST PATH PROBLEM WITH RE-ROUTING EN-ROUTE

Banu Karakaya

M.S. in Industrial Engineering

Supervisor: Asst. Prof. Osman Alp

August, 2008

In this study, we examine the shortest path problem under the possibility of "re-routing" when an arc that is being traversed is blocked due to reasons such as road and weather conditions, congestion, accidents etc. If an incident occurs along the arc being traversed, the vehicle either waits until all effects of the incident are cleared and then follows the same path thereafter, or returns to the starting node of that arc and follows an escape route to the destination node, the latter course of action is called as "re-routing". Also, we consider that this arc is not visited again throughout the travel along the network when an incident occurs and the alternative of not following this arc after the event is chosen. We propose a labeling algorithm to solve this specific problem. Then, a real case problem is analyzed by the proposed algorithm and several numerical studies are conducted in order to assess the sensitivity of the probability and travel time parameters.

Keywords: Shortest Path Problem, Routing, Re-routing, Shortest Path, Labeling Algorithm.

ÖZET

YOL ÜZERİNDE YENİDEN ROTALAMAYI DİKKATE ALAN EN KISA YOL PROBLEMİ

Banu Karakaya
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yard. Doç. Dr. Osman Alp
Ağustos, 2008

Bu çalışmada, üzerinden geçilmekte olan herhangi bir arkın, yol ve hava koşulları, tıkanıklık, kazalar vs. gibi sebepler nedeniyle kapanması durumunda "yeniden rotalama" ihtimali göz önüne alınarak, en kısa yol problemi incelenmiştir. Eğer geçilmekte olan ark üzerinde bir olay olursa, araç, ya olayın bütün etkilerinin te mizlenmesini bekler ve sonrasında aynı rotayı takip eder ya da olayın gerçekleştiği arkın başlangıç noduna geri döner ve varış noduna kadar başka bir kaçış rotasını takip eder. Sonuncu hareket tarzı "yeniden rotalama" olarak adlandırılır. Ayrıca, eğer bir olay olur ve olay sonrasında bu arkın takip edilmemesi alternatifi seçilirse, ağ üzerinde yolculuk boyunca bu arkın yeniden ziyaret edilmediği dikkate alınmıştır. Bu spesifik problemi çözmek için bir etiketleme algoritması önerilmiştir. Önerilen algoritma kullanılarak gerçek bir problem üzerinde analizler yapılmıştır. Yolculuk zamanı ve kaza olasılığı parametrelerinin duyarlılığını gözlemlemek amacıyla çeşitli sayısal çalışmalar yürütülmüştür.

Anahtar sözcükler: En Kısa Yol Problemi, Rotalama, Yeniden Rotalama, En Kısa Yol, Etiketleme Algoritması.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Asst. Prof. Osman Alp for his supervision, suggestions, guidance and most importantly his patience and trust during my graduate study. In all my academic study in Bilkent, I have always been felt comfortable to follow my graduate path with the strong help of my supervisor and my teachers even if my undergradute study is not Industrial Engineering but Statistics.

I would like to thank to Assoc. Prof. Bahar Yetiş Kara and Dr. Banu Yüksel Özkaya for showing of kindness to accept to read and review this thesis and their substantial comments and suggestions.

Also, I am very thankful to my friends in Bilkent IE department for their sincere friendship and morale support, especially Gülay Samatlı, Esra Aybar, İpek Keleş, Erdinc Mert, Nurdan Ahat, Nasuh Çağdaş Büyükkaramıklı, Ayşegül Altın, Mehmet Mustafa Tanrıkulu, Fazıl Pac and as well as to all of the friends that I failed to mention here. It would have been very hard without their friendship.

I would like to express my gratitude to my managers and my colleaques in MilSOFT Software Technologies Inc. and MilSOFT Information Communication Technologies (MilSOFT ICT) for their continous support and patience throug hout my graduate study.

Finally, I would like to express my deepest gratitude to my dearest Burcu, Uygar and Onat for their everlasting love and support. Without their love, I would never have finished this thesis.

To my family

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Routing is defined as the process of choosing paths in a network along which data, commodities and traffic are transferred. Routing is crucial for many kinds of networks such as telecommunication, computer and transportation networks. When we consider the routing application in a transportation network, routing a vehicle through the network from a given origin to a given destination requires determination of paths with the objective of optimizing a specified routing metric such as minimization of travel time, minimization of transportation cost, etc. The shortest path problem, which is one of the classical problems of operations research literature, is the problem of finding the shortest route between a given origin and a destination on a given network. The classical shortest path problem is first applied to solve problems having deterministic and time-independent arc attributes with a single objective function. This classical problem has been extended by researchers in many ways to include various considerations such as stochastic and time-varying arc attributes, multiple objectives, etc. In this thesis, we deal with an extension of the classical shortest path problem with stochastic

arc attributes and with an option of "re-routing" of the entity travelling on the network. Due to this re-routing element, we basically focus on transportation networks but our analysis may be applied different types of networks as well.

In many applications of routing problems on transportation networks, unexpected incidents may occur which result in the blockage of the road for through traffic and hence causing delays of the transit times. The probability of having an incident along the given network implies the probability of road blocking after the event. For example, accidents involving several vehicles, release of a hazardous material (hazmat) from a hazmat truck due to an accident or miscarriage, heavy weather conditions such as tornado, thunderstorms, etc. may cause the closure of roads for some time until the consequences of the incident are cleared. Other than these incidents, from time to time heavy traffic congestions lead to unreasonably long delays in the suburban areas of metropolitan cities. Some examples of road closure due to such incidents occurred on transportation networks can be given. For instance, a truck carrying a hazmat caused an accident occurred on TEM Highway, Turkey on June 25, 2007. The road that the hazmat vehicle traversed had to be closed after this terrible accident. Also, disasters such as flooding resulting from thunderstorms, tornadoes occurring in the state of Illinois in United States of America (USA), which is accepted as an important transportation hub because of its geographical location, had resulted road closures. For instance, closure in Illinois due to flooding, which was published at the website of Wisconsin Department of Transportation, had realized on July 12, 2008. Many road blocking examples have been encountered recently due to the flooding occurred on June 2008 in Midwestern U.S. such as the road blocking due to flooding on June 15, 2008 in Wisconsin, USA. Also, Interstate 80 was closed in Cedar County, Iowa, east of the Iowa City area, due to flooding from the Iowa

River and Coralville Lake on June 14, 2008. Such incidents and dynamically changing road and traffic conditions indicate that the transit times observed en-route has a probabilistic structure. In general, we define "road blocking" as a delay encountered on the road due to some reason, where there is an associated probability that the travelling vehicle encounters such an event while traversing an arc and the transit time of the arc differs depending on whether such an event has been encountered or not. After a road blocking resulting from an incident occurs on transportation networks , the traveller faces with the choice of two alternatives: waiting along the arc that the event has occurred until it is cleared or turning back of the starting node of that arc and finding a new route to follow thereafter until reaching to the destination node. The latter case is defined as "re-routing" in this study. Our aim is to incorporate such re-routing decisions into the route selection models and find routes that basically result with shorter expected lengths.

In recent years, "Online Routing", has been a very popular and widely implemented strategy with the rapid advancement of computer-based technologies such as geographic information systems (GIS), global positioning systems (GPS) and general packet radio service (GPRS) etc. Such information and communication technologies have provided real-time availability of information to be used in re-routing decisions which is thought to lead to more feasible and beneficial solutions in the routing problems. When real-time information on link travel times is available to the routed vehicle by the use of such technologies, adaptive route choices may then be identified en-route with this information. Also, we may observe that the events that cause the road blocking in transportation networks problems have been monitored and reported via those technologies. If such an event occurs along the path that is chosen before the trip has been reported, the

vehicle may be directed to a new route by a central authorithy.

In the classical online routing problems, the path is chosen based on a predetermined criteria such as minimization of the (expected) length of the route, then adaptive routing strategies are generated en-route as the travel times and road conditions are revealed on the path. Online routing strategy may be implemented much more effectively if the events that have a certain likelihood to be observed along the path (road blockages, delays in transit times) and the possible re-routing decisions in case such events occur could be embedded in the route selection phase. Specifically, if the possibility of re-routing en-route is not considered when the path is being chosen, then the decision maker seeks a path that minimizes a performance measure. In such a case, even if the decision maker considers the probabilistic nature of the arc attributes, he/she can find a path that minimizes the expected measure, at best. Hence, there is no guarantee how that 'optimal' path will 'perform' if and when the vehicle needs to be re-routed en-route. For example, the path chosen could send the vehicle to a devious path (because its expected measure is the least) which has rare connections to escape routes. If one of the arcs on this path is blocked due to an adverse weather condition along that path, then the vehicle may not find a way to change its route and may get stuck on the road. Hence, re-routing can not be implemented even if it is technologically feasible. However, if the decision maker has routed the vehicle to a path around numerous escape routes, then the re-routing possibility could have been utilized much more effectively and the total expected measure could have been lowered. The main aim of this study is to show that choosing the path by considering this phenomena results in paths having lower expected measures when re-routing is an option en-route. This phenomena is especially important for hazardous materials (hazmat) transportation. When a road is blocked due

to some reason and a hazmat truck gets stuck on that road, the risk imposed to the environment could be considerably high since the population exposed to the risk while waiting for that road to be cleared would be high. So, it would be reasonable to select paths with more connections to escape routes in order to minimize the risk and consequences of such an event. Another crucial application area could be for routing vehicles after the occurrence of a disasterous event such as an earthquake, a flood or hurricane. Some of the roads would be unusable but there would be no perfect information about the conditions of the roads. In such cases, one could associate probability of roads being unusable depending on the physical and geographical characteristics of the incident and select a path with minimum expected duration, but such paths would be useless if it does not have sufficient connections to escape routes in case some arcs on the selected path are unusable.

Our problem is to find the shortest path that minimizes the expected value of a performance measure, specifically travel time, under the possibility of re-routing when an arc that is being traversed is blocked. Assuming that blocking occurs in a probabilistic manner due to the reasons such as road conditions, weather conditions, congestion, release of a hazmat etc., we consider the probabilities of such events in our routing problem. In this study, we assume that the probabilities of road blocking and not blocking and their corresponding arc travel times are known. We also assume that if an arc that is being traversed is blocked, the re-routing alternative is evaluated instantly, and either the vehicle waits till the blocked road is cleared or a new escape route is selected and followed thereafter to the destination node after the road blocking. Moreover, we assume that an arc that is left due to blocking is not visited again in the network. That is to say, this arc is removed from the network when the accident occurs and the alternative

of not following this arc after the event is chosen. In this study, we propose a labeling algorithm to solve the specified problem and aim to demonstrate the advantage of this specific re-routing model.

The remainder of the thesis is organized as follows: In this chapter, we mention about a very simple example which is used to motivate our problem. Then, in Chapter 2, some background information about Routing and Online Routing subjects will be introduced. The formulation of the re-routed shortest path problem considering road blocking will be presented in Chapter 3. Also, model and suggested algorithm will be mentioned in this chapter. In Chapter 4, computational study of the proposed algorithm will be given. Analysis performed with the real world example will also be explained in this chapter. Finally, in Chapter 5, we conclude the thesis by summarizing the overall study and making some remarks about the related work.

1.1 Motivating Example

In this section, we motivate our problem with a toy example. Consider a network with a given origin and a destination. Specifically, we have an undirected graph $G = (N, A)$ having $N = 6$ nodes and $A = 7$ arcs. The source node, S, is represented as node 0 and the destination node, D, is represented as node 5 in our example. We assume that there may occur incidents on some arcs resulting with a blockage on that arc along the given network. Hence, each arc (i, j) has a pre-assigned probability, p_{ij}, of being blocked while the vehicle traverses that arc. Also, due to the occurrence of such events, we assign two different travel times to each arc such as travel time of the arc provided that it is not blocked, d_{ij}^{U}, and

travel time of the arc provided that it is blocked, d_{ij}^B. For simplicity, we assume that there may occur at most $k = 1$ incident along the network throughout the travel. The parameters of the sample example are summarized in the Table 1.1. The illustration of our network is seen as in the Figure 1.1.

Table 1.1: Parameters of Motivating Example

Arc (i, j)	p_{ij}	d_{ij}^U	d_{ij}^B
(0, 1)	0.2	1.0	2.0
(0, 3)	0.3	1.0	8.0
(1, 2)	0.2	2.0	4.0
(2, 5)	0.2	1.0	3.0
(3, 4)	0.1	2.0	3.0
(3, 5)	0.1	2.0	8.0
(4, 5)	0.1	4.0	8.0

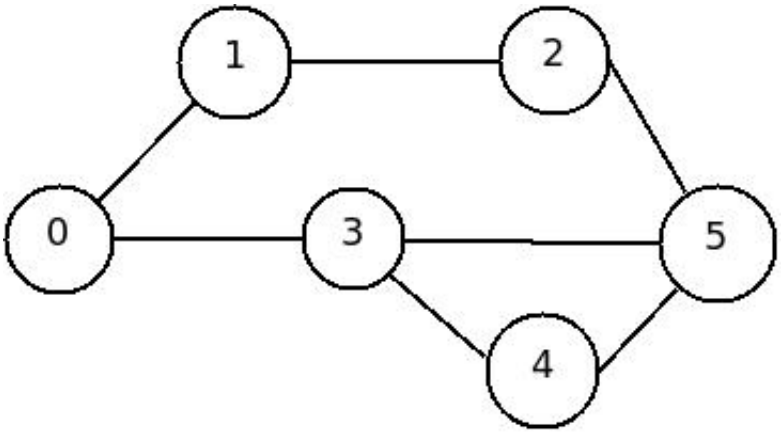

Figure 1.1: Network of Motivating Example

Our objective is to find an optimal route having the least expected travel time along the network. We have 3 possible paths that can be the optimal solution, these are;

$$\text{Path } 1: 0-1-2-5$$

$$\text{Path } 2: 0-3-5$$

$$\text{Path } 3: 0-3-4-5$$

If the re-routing option is not considered while choosing a path, then one should find the path with least expected duration. For each of the paths above, one can simply calculate the expected duration and select the minimum. For our example, the optimal path is Path 1, $0-1-2-5$, with the least expected travel time of 4.776 minutes. This is the solution of shortest path considering expectations and no re-routing alternative. Now let us evaluate the expected travel time for each path assuming that the truck will be re-routed en-route if at most $k = 1$ accident may occur along the network. While solving the problem with the option of re-routing, we assume that the arc that is blocked due to an incident is removed and is not considered again in the network. So, we calculate the expected travel time of the possible paths by regarding this issue. For example, if we do not have any incidents throughout the travel, the expected travel time of Path 1 is $(0.8*1+0.8*2+0.8*1) = 3.2$. However, if we have an incident along the arc $(0, 1)$ on Path 1, we either choose to wait along that path until the effects of the incident are removed or we choose not to wait and so go back to the node 0 and continue from that node to the destination node. If we choose to wait, the total expected travel time is $0.2 * (2 + 2 + 1) = 1.0$; if we do not wait, the total expected travel time is $0.2 * (1 + least expected travel time of going from node 0 to node 5) =$ $0.2 * (1 + 1 + 2) = 0.8$. In this case, we point out that while calculating the

expected travel time from node 0 to node 5, we remove the arc $(0, 1)$ in which the incident has occurred and calculate other possible paths from node 1 to the destination node. Similarly, if we have an incident along the arc $(1, 2)$ on Path 1 and we choose to wait, the total expected travel time is $(1 + 0.2 * (4 + 1)) = 2.0$. If we do not wait, the expected travel time is $(1 + 0.2 * (2 + 1 + 1 + 2)) = 2.2$. For Path 1, if we have an incident along the arc $(2, 5)$ and we choose to wait, the total expected travel time is $(1 + 2 + (0.2 * 3)) = 3.6$. If we do not wait, the expected travel time is $(1 + 2 + 0.2 * (1 + 2 + 1 + 1 + 2)) = 4.4$. Hence, the least expected travel time for Path 1 is obtained as 4.4 minutes, when we consider re-routing en-route option. By this methodology, we calculate the expected travel times for Path 2 and Path 3. We obtain the least expected travel time as 3.95 for Path 2, $0 - 3 - 5$, when re-routing en-route is considered. Hence, we see that there exists $((4.4 - 3.95)/3.95 * 100) = 11.39\%$ reduction in the overall travel time when the option of re-routing en-route is considered. Also, we observe that there is an improvement in the optimal paths of the problems such that they differ from each other. That is, if we do not consider the re-routing option, we have Path 1, $0 - 1 - 2 - 5$, as the optimal path among the other paths. However, if we consider the re-routing option, Path 3, $0 - 3 - 5$ along which has possible escape routes is the optimal path that we obtain in our example. Thus, we may conclude that the re-routing option enables the travelers to follow some escape routes along the given network if there is an incident occurring on the network. This can be resulted from the case that if there is an incident along the arc $(3, 5)$, we choose going back to the starting point of the arc, node (3), and continue from that arc to the destination node by considering the other paths in the network as an alternative. This course of action shows "re-routing" as specified in our definition of the problem.

This simple example demonstrates that considering the option of re-routing may result in better solutions in some cases. Also, re-routing would be desirable if the path chosen is a devious path and the vehicle may get stuck on any arc along the chosen path when an incident occurs throughout the travel.

Chapter 2

LITERATURE REVIEW

In this chapter, we first review the literature on the routing problems with stochastic arc attributes having single and multiple objective functions. While citing about these problems, we explain the studies performed in hazardous material transportation which is one of the most considered application areas of routing. Then, we present the literature about the online routing problems in transportation networks and applications performed in real world examples.

In literature, routing problems have been considered as one of the classical shortest path problem, and the classical shortest path algorithms have been developed and used to solve these problems. Different variants of routing problems have been widely studied by many researchers. The problems can be grouped considering the characteristics of the link attributes (travel time or cost) as deterministic or stochastic, and type of objective function as single and multiple. Moreover, many problems in this field have been raised as the combination of

these two groups. Most of the researchers worked with the deterministic networks, where the arc attributes are certain. The computationally most efficient algorithm for such networks was first developed by Dijkstra [5]. However, when considering the routing of a vehicle from an origin to a destination, it is more likely that the travel times along the trip are random rather than deterministic, which results the consideration of stochastic arc attributes (i.e. travel times) in the literature. Hall [12] studied the shortest path problem in a transportation network where the arc travel times are random and vary with time of day to find the least expected travel time path between two nodes. He firstly pointed out that the classical shortest path algorithm like Dijkstra's proposed may fail to find the least expected travel time path on networks with random time-dependent travel times. He proposed an algorithm based on dynamic programming to solve this kind of problem. List et al. [13] presents a survey of routing/scheduling in hazardous materials transportation where stochastic arc attributes are considered. Wijeratne et al. [23] propose a solution methodology to select the non-dominated routes in a network having the stochastic arc attributes. They applied their algorithm in hazardous materials transportation. Fu and Rilett [9] show that the stochastic networks with time-dependent travel times can not be solved exactly using the standard shortest path algorithms, and they develop a heuristic algorithm based on k shortest path algorithm to solve the defined problem. They approximate the mean and variance of the arc travel times by developing models so that they result in improved solutions. A stochastic, time varying transportation model is presented by Miller-Hooks and Mahmassani [16]. The paper considers the selection of routes in the network where the travel time and risk measures are not known with certainty. Many solution procedures are presented in their work. Cheung [4] considers the problem of finding a shortest path in a network with

discrete positive and independent random arc costs where the arc costs are realized over time. When a node i is visited, the actual arc cost of going out from this node become known, and a shortest path is changed accordingly. He proposes a routing policy to dynamically form a shortest path in the specified network by using the idea of the classical label-correcting algorithm. Erkut and Tjandra [7] studied the stochastic time varying networks with en-route decisions and data updates based on real-time information. They used the results of Fu and Rilett [9] and proposed a procedure based on a label-correcting algorithm to generate nondominated paths. Their procedure includes a structure to update estimations of travel times and costs when the real-time information is gathered en-route. Another study on stochastic networks with time varying arc attributes is performed by Chang et al. [3]. They developed a multicriterion a priori route choice in stochastic time varying networks where travel times are normally distributed. They proposed an algorithm to find non-dominated paths in such networks and tested their algorithm on smaller and larger networks to see the effects of parameter changes. In a working paper written by Nielsen et al. [17], they consider bicriterion a priori route choice problem in stochastic time varying networks. A new algorithm based on two-phase approach has been proposed to find the set of efficient paths with the objective of minimizing both expected time and cost. They tested the algorithm on test instances considering both the expectation and min-max criteria. Opasanon and Miller-Hooks [18] study the stochastic networks having time varying arc attributes with multiple criteria. They present an exact algorithm to find solution strategies considering the importance of multiple criteria at each node where the travel times are experienced. They allow selecting the best path according to the preferred criterion when the real travel times are observed. Some research papers are also presented in the literature where routing

problems with stochastic arc attributes are mentioned.

In recent years, a new research topic called dynamic routing or online routing, has been growing rapidly since computer-based technologies such as geographic information systems (GIS), global positioning systems (GPS), general packet radio service (GPRS) etc. have considerably improved the possibilities for dynamic routing. The improvements in information and communication technologies have provided real-time availability of information to be used in distribution logistics, which leads to more feasible and beneficial solutions in the routing problems. When real-time information on link travel times is available to the routed vehicle, adaptive route choices may then be identified en-route with this information. Thus, availability and use of the real-time information as the routed vehicle traverses its path has lead researchers to study extensively this new type of problem area. Psaraftis [21] reviews the literature on dynamic routing and its relation with computer-based technologies. Also, real-time vehicle routing literature is reviewed by Ghiani et al. [11]. They mention about the basic definitions, concepts and algorithms in existing real-time vehicle routing environment. Blue et al. [2] discuss the real-time bi-objective route choice for in-vehicle route guidance systems. They describe a new idea called trip quality which is composed of two objectives as minimizing travel time and trip complexity. A simulation modeling approach is used to show the effects on the trip performance of pretrip and dynamic routing strategies. Miller-Hooks [14] studies the stochastic time varying networks to select the minimum expected time paths in an adaptive way. She shows that one can make improved routing decisions when the travel times are realized en-route once the arc is traversed. This is called as adaptive selection in the paper. Since the optimal path depends on the real-time information of travel times obtained en-route, a single path may not be adequate so that a set of

strategies are constructed to determine the minimum expected time paths. An efficient algorithm based on label-setting procedure is proposed to find the adaptive least expected time paths. She concludes that an adaptive selection procedure can lead to improved routing decisions over a priori path selection. As Miller-Hooks study, Fu [8] develops an optimal adaptive routing algorithm in networks in which link travel times are modeled as random variables with known mean and standard deviation. Realizations of them can be estimated based on the real-time information of travel times gathered from the in-vehicle route guidance system. A labeling algorithm was proposed to solve the specified problem. The efficiency of the proposed algorithm and advantages of adaptive routing systems are represented in the paper. Also, Miller-Hooks and Mahmassani [15] examine the differences of paths selected for a priori and time-adaptive decisions in stochastic time varying networks. Two dominance criteria are defined for the comparison of two paths over a time period, their application to compare the paths depends on the problem area. Gao and Chabini [10] establish a theoretical framework for optimal routing policy (ORP) problems in stochastic time-dependent networks. They provide a comprehensive taxonomy of the ORP problems considering the stochastic dependency and real time availability of data. They thoroughly discuss some variants of the problem in their paper. Also, they develop an operational algorithm to solve ORP problem with perfect online information, i.e. realizations of all link travel times up to current time period are known.

The most related work encountered in the literature is about shortest path problems with recourse. The first study is performed by Andreatta and Romeo[1]. They consider a stochastic network with static arc attributes. In their study, the decision maker is assumed to learn the status of an arc that will be traversed when the decision maker reaches the node from which the arc comes from. If

he learns that the next arc is inactive, he can decide rerouting. They provide a stochastic dynamic programming formulation to solve the problem. They firstly state that the complexity of the algorithm can grow exponentially with the number of arcs. They study the restricted version of the problem and show that polynomial algorithms exist for those versions. Our work is somewhat different from their's in such that we assume re-routing may be implemented as traversing the arc. We see this as the main contribution in the literature. In this thesis, we also try to improve the objective function of the stochastic dynamic problems by generating paths that allow re-routing options en-route. Polychronopoulos and Tsitsiklis[19] study the stochastic dynamic networks under the assumption that the realization of the arc costs is learned progressively, as the graph is traversed. When more link travel time realizations are learned, it is said that the network becomes closer to a deterministic one. They propose two alternative models in which the arc costs are modeled as independent or dependent random variables, respectively. For these two models, they develop dynamic programming algorithms to solve the problem. For the model having the arc costs as dependent variables, they observe that the dynamic programming algorithm developed is exponential in the number of possible realizations of arc costs. Also, for the second model, their algorithm is exponential in the number of arcs. In their study, the cost of an optimal policy for two models is compared with the cost resulting from the application of a heuristic policy. They conclude that modeling uncertainty in these type of problems is vital. Some of the main concepts are originated from this study. However, our work differs from their's as such that re-routing is implemented while traversing the arc, not at the time that an end node of that arc is visited. Waller and Ziliaskopoulos[22] consider the problem in a static network with limited spatial and time arc cost dependencies. They

assume that no further information is obtained through spatial dependence given the cost of predecessor arcs. This limited dependence is modeled by distributions of the cost of an arc, conditioned on the cost of each one of its upstream arcs. To obtain understanding about the online shortest path problems, they study the relationship of the a priori versus online solution. Efficient algorithms are developed for special cases of the problem. Different from the works of Cheung[4] and Polychronopoulos and Tsitsiklis[19], Provan[20] studies the problem of finding a shortest path in a directed network in which the link travel times can be dependent. In his study, the realization of arc lengths are revealed only upon arrival at the tail of that arc. Also, each time an arc is traversed, it is assumed that the link travel times take new independent values, that is to say they are after each arc traversal. This "reset assumption" is firstly defined by Provan[20] in the literature of online stochastic shortest path problems. An algorithm is proposed to solve the given problem in the presence of this assumption, and a new case with negative arc lengths is studied in his work. In this study, we analyze the online shortest path problem with stochastic arc attributes when re-routing is an option en-route. The main contribution of our work is that the model and algorithm of the problem have been developed provided that the realizations of arc travel times are revealed while traversing the arc. A labeling algorithm is proposed to solve the specified problem and to demonstrate the advantage of this specific re-routing model in our study.

Chapter 3

PROBLEM DEFINITION AND SUGGESTED ALGORITHM

In this chapter, we pose the "shortest path prolem with re-routing en-route" and present a labeling algorithm that can be used to solve this problem. In Section 3.1, problem definition is explained in detail. Then, associated notation, network description and main assumptions of the problem are presented in Section 3.2. Section 3.3 presents the algorithm. In Section 3.4, a simple example is solved by applying the proposed algorithm in order to show the execution of the algorithm.

3.1 Problem Definition

In this study, we consider the problem of finding the shortest path from a given origin to a given destination when the vehicle traversing the path is allowed to

change its route in case an arc is blocked en-route. Our main aim is to incorporate the possible re-routing decisions to be implemented en-route, in the selection process of the path in terms of a performance measure, e.g. expected path duration. Note that, it may be desirable to change the route if the expected waiting time until the road is cleared is unreasonably long. By means of road blocking, we refer to as having an "incident" along the arc that is being traversed in a probabilistic manner due to the reasons such as a traffic accident, road conditions, weather conditions, security threats due to release of a hazardous material or fire, irregular congestion, etc. We refer to all of these accidents simply as an 'incident'. When an incident occurs along the arc being traversed, we say that the arc is "blocked". On a blocked arc, the vehicle either waits until all effects of the incident are cleared and then follows the same path thereafter, or returns to the starting node of that arc and follows an escape route to the destination node, i.e. the vehicle is re-routed. When an incident occurs on the arc being traversed along the network (i.e. when the arc is blocked), we assume that this arc will not be visited again if re-routing alternative is chosen. This is because of the fact that the arc being blocked will be very congested for some time after the incident. Thus, we consider to follow the subpaths along the network which the arc being blocked is removed for the rest of the travel. Note that an incident may occur at any point of an arc. Hence, after starting traversing a blocked arc (without knowing that it is blocked), the truck driver may notice that the road is blocked at any point on that arc. Indeed, a possible re-routing decision can be given at any point on the arc depending on the starting point of the road blockage. However, we assume that the re-routing decision is given right on the middle of the arc, on the average. So, we consider that the travel is constituted from going forward until the middle of the arc and then backward to the starting

point of that arc, if the re-routing option is chosen. Thus, the duration of the travel in this case equals the total travel time of the arc (i, j) provided that it is not blocked. Also, we assume that every arc on the network permits a possible escape by driving the truck back to the starting point. If there are some arcs that do not permit this on the network, due to some physical constraints, our algorithm should be modified by forcing the "waiting" alternative only for such arcs. This modification can easily be adapted to our study.

The problem is to find the shortest path that minimizes the expected value of a performance measure or metric such as travel time, exposed risk, cost of traveling, etc. In this study, the objective function is considered to be the minimization of the expected travel time, nevertheless any other metric can easily be adopted. The possibility of re-routing implies the possibility of an incident along the arcs of the specified network. Hence, in our environment, each arc is associated with a probability of being blocked, or equivalently one of the afermentioned incidents being occurred. As expected, the duration of traversing the arc after an incident occurs would be more than that of traversing the arc without any incident. Thus, we assume parameters for the probability of observing an arc being blocked and assume corresponding durations for the arc depending on whether the arc is blocked or not. Finally, we assume that there may occur at most K incidents along the network throughout the travel where K is a given parameter.

3.2 Network Description

We consider a traffic network represented by an undirected graph $G = (N, A)$ consisting of a finite set of nodes N and arcs A. In this specified network, we have an origin and a destination. We denote the origin and destination nodes by S and D, respectively. We assume that there may occur at most K incidents until reaching to the destination from the origin. Each arc (i, j) in G has an associated probability of having an incident (being blocked) along the arc, denoted by p_{ij}, provided that K incidents in total have not been observed yet. If the entity have already observed K incidents en-route, then the probability that the successive arcs on its path will be blocked becomes zero. The travel time of the arc (i, j) if that arc is blocked and not blocked are represented as d_{ij}^B and d_{ij}^U, respectively. The notation used in our model is summarized as in the following table:

Table 3.1: Summary of Notation

N	Set of nodes of the original network
A	Set of arcs of the original network
S	Origin Node
D	Destination Node
p_{ij}	probability of the arc (i, j) being blocked while traversing it provided that at most $K-1$ incidents have occurred on the path since the origin node up to node i
d_{ij}^U	travel time of the arc (i, j) provided that it is not blocked
d_{ij}^B	travel time of the arc (i, j) provided that it is blocked
K	Maximum number of incidents that can occur during the travel from origin to destination
$[ij\|G]^k$	The total expected travel time from node i to j on the Network G provided that at most k arcs may be blocked on G
$[G\backslash(i, j)]$	A sub-network obtained by excluding the arc (i, j) from the original network G
T	Set of temporary nodes
P	Set of permanent nodes
C	Active node
$Label(i)$	Current label of node i

3.3 Algorithm

In this section, we present an algorithm that is develeoped for solving the specified problem. The algorithm assumes that at most K incidents may occur along the path that the truck traverses (including all arcs visited by possible re-routing decisions) until reaching the destination node. Here K is a given parameter. The main algorithm iterates through $k \in \{0, 1, ..., K\}$ where in each iteration the subroutine, $k - Acc(G)$ is executed. This subroutine solves the main problem on a network G by assuming that at most k incidents may occur along the path and by using the optimal solutions generated by $(k-1) - Acc(\cdot)$ routines that were solved in the previous iteration. The algorithm can be represented as follows:

Main Algorithm

1. Let $k = 0$
2. Let S be set of all subsets of A with exactly $K - k$ elements.
3. For all $s \in S$
   Solve $k - Acc(H \backslash s)$ and find $[iD|H \backslash s]^k$ for $i \in H \backslash s$
4. Let $k = k + 1$ If $k < K$ then go to Step 2 else stop.

where

k-Acc(H) algorithm:

Step 1 $Label(i) = \infty$ for all $i \in H$, $P = \emptyset$ and let T be the set of all nodes of H.

Step 2 Let $[DD|H]^k = 0$ and $Label(D) = 0$. Remove D from T and insert it in P. Let P =D

Step 3 Let $j = D$.

Step 4 Do for all $i \in T$ such that $(i, j) \in H$ where j is the active node:

a Calculate the expected travel time between i and j as

$$[iD|H]^k = (1 - p_{ij}) \left(d_{ij}^U + [jD|H]^k\right) + p_{ij} \min \left\{d_{ij}^B + [jD|H]^{k-1}, d_{ij}^U + [iD|H \backslash (i,j)]^{k-1}\right\}$$

b $Label(i) = \min\{Label(i), [iD|H]^k\}, [iD|H]^k = Label(i)$

Step 5 Let j be an active node having $j = \arg\min\{Label(i) : i \in T\}$. Remove j from T and insert it in P.

Step 6 If $T = \emptyset$ then stop. Otherwise go to Step 4

This algorithm is a labeling algorithm. The labeling measure is the expected travel time between node i and node D. In other words, at any given iteration of the algorithm, the label of an arc stores the minimum expected travel time (duration) found so far to reach from that node to the destination node. In the main algorithm, first of all, we start with the case where $k = 0$, which is the base case of the problem. When there is no incident in the network, we solve the classical shortest path algorithm with the arc travel times having the value of not blocking case, i.e. d_{ij}^{U}. Then, iteratively we solve $k - Acc(H)$ problem. The algorithm for $k-Acc(H)$ problem is executed as follows: We first assign the labels of all nodes in the network G as infinity and put them in the temporary node set, T. Then, we set the permanent node set, P, as empty at the starting point. Now, we consider the destination node, D, where the total expected travel time from node D to D is 0. We define the label of D as 0. After doing this, we remove the destination node from the temporary node set and define it as permanent node. So, we let $P = D$. Then, we assign the destination node D as the active node j since its label is the minimum of all the other temporary nodes in G. Active node is defined as the temporary node whose label is the minimum among the other temporary nodes. When a node is set as the active node, it is removed from the temporary node set, T, and inserted in P. This can be done since there is no other node from where it can be reached to the destination node in a shorter way because of the travel times being nonnegative. In the temporary node set, we find the nodes that are connected with an arc to the active node j and we calculate the expected travel time between those nodes and the destination node, one by one, by using the equation of Step 4.a of the algorithm. This equation consists of two parts considering the possibility of road blocking. While traversing the arc (i, j), we encounter one of the following two options: either the road is blocked

or not. If the road is not blocked during the traversal, we reach from node i to node j with the travel time d_{ij}^U. Then, we continue from node j towards the destination node, D, with the minimum travel time of reaching from node j to D with at most k incidents. This is represented by the permanent label of node j, $[jD|H]^k$ While calculating the expected travel time in this case, we add two values, d_{ij}^U and $[jD|H]^k$, then multiply the result with $(1 - p_{ij})$, the probability of the arc (i, j) being not blocked. However, if a road blocking occurs on the arc (i, j), we need to make a decision as to whether wait until the road blockage is cleared and continue from node j to the destination node, D, or return back to node i and find another route to reach to node D. If we select the former course of action, then we spend d_{ij}^B time units on arc (i, j). After reaching node j, we consider the minimum travel time of reaching from node j to node D with at most $(k - 1)$ incidents since one incident has already been observed on the arc (i, j). Hence, this possibility brings an expected travel time of $(d_{ij}^B + [jD|H]^{k-1})$. If we choose to re-route and return back to node i, then we spend d_{ij}^U time units and need to consider reaching from node i to the destination node D with at most $(k - 1)$ incidents. Due to our assumption, we do not consider the arc (i, j) in calculating this value. Hence, this possibility brings an expected travel time of $(d_{ij}^U + [iD|H\backslash(i, j)]^{k-1})$. Then, we take the minimum of these two values and find the best decision to make, in case the arc (i, j) is blocked. Note that, the minimum expected travel time of reaching from node i or from node j to node D are already calculated in the previous iteration (k-1st iteration) of the algorithm. When k is incremented by one, we will need the minimum expected travel time of reaching from each node to the destination node when one of the arcs on the original network is blocked. Hence, we need to solve k-acc $[G\backslash(i, j)]$ algorithm for all $(i, j) \in A$. After finding the minimum value, we multiply the result with the

probability of the arc (i, j) being blocked. Finally, we add the results of the two cases to find the expected travel time of our problem according to the equation of Step 4.a of the algorithm. Then, after calculating the expected travel time of going from node i to the destination node for each node in T, we update the labels of the nodes in T as the number found by the minimum operator in Step 4.b. From this point on, we consider that the expected travel time between those nodes and the destination node is this updated value. Next, we compare all the labels of nodes in T and find the minimum of them. This node is set as the active node, j, and it is removed from the temporary node set and inserted in the permanent node set, as stated in the definition of the active node. We continue with the algorithm by applying the same logic defined above. In the execution of the algorithm, if the temporary node set equals to the empty set, we terminate the algorithm.

From the definition of our algorithm's steps, the complexity of the algorithm is specified by considering the complexity of $k - Acc(H)$ problem, which is order of number of nodes times number of arcs. The algorithm's complexity is then order of combination of number of arcs with number of incidents times the complexity of $k - Acc(H)$ problem. That is, the complexity is exponential in the number of arcs and the number of incidents.

3.4 A Simple Example

In this section, we solve the motivating example stated in Chapter 1 by applying our algorithm in order to explain how the algorithm is executed. Consider $K = 1$.

k-Acc(H) algorithm:

Step 1 $Label(0) = Label(1) = Label(2) = Label(3) = Label(4) = Label(5) = \infty$ for all $i = 0, 1, 2, 3, 4, 5 \in H$,

$P = \emptyset$ and let $T = 0, 1, 2, 3, 4, 5$ be the set of all nodes of H.

Step 2 Let $[55|H]^1 = 0$ and $Label(5) = 0$. Remove 5 from T and insert it in P. Let P =5

Step 3 Let $j = 5$.

Step 4 Do for all $i \in T = 0, 1, 2, 3, 4$ such that $(i, 5) \in H$ where 5 is the active node:

a Calculate the expected travel time between i and 5 as

$[25|H]^1 = (1 - p_{25}) \left(d^U_{25} + [55|H]^1\right) +$

$p_{25} \min \left\{d^B_{25} + [55|H]^{1-1}, d^U_{25} + [25|H \backslash (2, 5)]^{1-1}\right\}$

$[25|H]^1 = ((1 - 0.2) * (1 + 0)) + 0.2 * min(3 + 0), (1 + 6) = 0.8 * 1 + 0.2 *$

$min(3, 7) = 1.4$

$[35|H]^1 = (1 - p_{35}) \left(d^U_{35} + [55|H]^1\right) +$

$p_{35} \min \left\{d^B_{35} + [55|H]^{1-1}, d^U_{35} + [35|H \backslash (3, 5)]^{1-1}\right\}$

$[35|H]^1 = ((1 - 0.1) * (2 + 0)) + 0.1 * min(8 + 0), (2 + 5) = 0.9 * 2 + 0.1 *$

$min(8, 7) = 2.5$

$[45|H]^1 = (1 - p_{45})\left(d^U_{45} + [55|H]^1\right) +$

$p_{45} \min\left\{d^B_{45} + [55|H]^{1-1}, d^U_{45} + [45|H\backslash(4,5)]^{1-1}\right\}$

$[45|H]^1 = ((1 - 0.1) * (4 + 0)) + 0.1 * min(8 + 0), (4 + 4) = 0.9 * 4 + 0.1 *$

$min(8, 8) = 4.4$

b $Label(2) = \min\{Label(2), [25|H]^1\} = 1.4,$

$Label(3) = \min\{Label(3), [35|H]^1\} = 2.5,$

$Label(4) = \min\{Label(4), [45|H]^1\} = 4.4$

Step 5 $min(Label(2), Label(3), Label(4)) = Label(2)$

Let $j = 2$ Remove 2 from T and insert it in P.

$T = 0, 1, 3, 4$ and $P = 5, 2$.

Step 6 Since T is not $\emptyset$, go to Step 4.

Step 4 Do for all $i \in T = 0, 1, 3, 4$ such that $(i, 2) \in H$ where 2 is the active node:

a Calculate the expected travel time between i and 2 as

$[15|H]^1 = (1 - p_{12})\left(d^U_{12} + [25|H]^1\right) +$

$p_{12} \min\left\{d^B_{12} + [25|H]^{1-1}, d^U_{12} + [15|H\backslash(1,2)]^{1-1}\right\}$

$[15|H]^1 = ((1-0.2)*(2+1.4)) + 0.2 * min(4+1), (2+4) = 0.8 * 3.4 + 0.2 *$

$min(5, 6) = 3.72$

b $Label(1) = \min\{Label(1), [15|H]^1\} = 3.72.$

Step 5 $min(Label(0), Label(1), Label(3), Label(4)) = Label(3)$

Let $j = 3$ Remove 3 from T and insert it in P.

So, $T = 0, 1, 4$ and $P = 5, 2, 3$.

Step 6 Since T is not $\emptyset$, go to Step 4.

Step 4 Do for all $i \in T = 0, 1, 4$ such that $(i, 3) \in H$ where 3 is the active node:

a Calculate the expected travel time between i and 3 as

$$[05|H]^1 = (1 - p_{03}) \left(d^U_{03} + [35|H]^1\right) +$$

$$p_{03} \min \left\{d^B_{03} + [35|H]^{1-1}, d^U_{03} + [05|H \backslash (0,3)]^{1-1}\right\}$$

$$[05|H]^1 = ((1-0.3) * (1+2.5)) + 0.3 * min(8+5), (1+4) = 0.7 * 3.5 + 0.3 *$$

$$min(13, 5) = 3.95$$

$$[45|H]^1 = (1 - p_{43}) \left(d^U_{43} + [35|H]^1\right) +$$

$$p_{43} \min \left\{d^B_{43} + [35|H]^{1-1}, d^U_{43} + [45|H \backslash (4,3)]^{1-1}\right\}$$

$$[45|H]^1 = ((1-0.1) * (2+2.5)) + 0.1 * min(3+5), (2+4) = 0.9 * 4.5 + 0.1 *$$

$$min(8, 6) = 4.65$$

b $Label(0) = \min\{Label(0), [05|H]^1\} = 3.95,$

Step 5 $min(Label(0), Label(1), Label(4)) = Label(1)$

Let $j = 1$ Remove 1 from T and insert it in P.

So, $T = 0, 4$ and $P = 5, 2, 3, 1$.

Step 6 Since T is not $\emptyset$, go to Step 4.

Step 4 Do for all $i \in T = 0, 4$ such that $(i, 1) \in H$ where 1 is the active node:

a Calculate the expected travel time between i and 1 as

$[05|H]^1 = (1 - p_{01}) \left(d_{01}^U + [15|H]^1\right) +$

$p_{01} \min \left\{d_{01}^B + [15|H]^{1-1}, d_{01}^U + [05|H \backslash (0,1)]^{1-1}\right\}$

$[05|H]^1 = ((1 - 0.2) * (1 + 3.72)) + 0.2 * min(2 + 3), (1 + 4) = 0.8 * 4.72 +$

$0.2 * min(5, 5) = 4.776$

b $Label(0) = \min\{Label(0), [05|H]^1\} = 3.95,$

Step 5 $min(Label(0), Label(4) = Label(0))$

Let $j = 0$ Remove 0 from T and insert it in P.

So, $T = 4$ and $P = 5, 2, 3, 1, 0$.

Step 6 Since T is not $\emptyset$, go to Step 4.

Step 4 Do for all $i \in T = 4$ such that $(i, 0) \in H$ where 0 is the active node:

Since no such arc exists, we skip these steps for that case.

Step 5 $minLabel(4) = Label(4)$ Let $j = 4$ Remove 4 from T and insert it in P.

So, $T = \emptyset$ and $P = 5, 2, 3, 1, 0, 4$.

Step 6 Since $T = \emptyset$, we terminate the algorithm.

Hence, we find the least expected time as 4.776 that is same as we have found in Chapter 1. Also, if we solve the problem for $K = 2$, we obtain the least expected travel time considering no re-route option as 4.984 min. If the option of re-routing en-route is considered, the least expected travel time is obtained as 4.253 min. The optimal paths differ such that Path 1 is optimal if re-routing option is not considered, and Path 3 is optimal if re-routing option is considered en-route.

Chapter 4

COMPUTATIONAL STUDIES

In this chapter, we examine the computational performance of our algorithm by performing studies based on a real example and also some different versions of our problem. Our aim is to demonstrate how our algorithm performs on a real case problem and is to compare the results obtained from the different types of the problem. In the first section, we mention characteristics of the real world problem and the solution produced by performing our algorithm is to be evaluated. Then, we quantitavely analyze the results obtained from different versions of the problem in the subsequent section to obtain some insights about the algorithm.

4.1 Real Case Analysis

We use the network specified in the paper of Erkut and Alp [6] as a real world example. This specified network represents the interstate highway network in the Northeastern United States. It has 138 nodes and 368 arcs. For our problem,

we use the travel duration (in minutes) given in the network exactly as it is for travel time of the arc (i, j) provided that it is not blocked, d_{ij}^U. We then generate randomly the travel times of the arc (i, j) provided that it is blocked, d_{ij}^B, by multiplying d_{ij}^U time units by some specified parameter. That is, we assume that d_{ij}^B's are between $(5.2{*}d_{ij}^U)$ and $(5.6{*}d_{ij}^U)$. For each arc, we also randomly generate the probability of the arc (i, j) being blocked while traversing it by assuming that the probability is less than 0.5. Given this specific network, we solve for the problem under the assumption that there may occur at most $k = 1$ incident along the network throughout the travel. By applying our algorithm, the expected travel time from the source node to the destination node is found as 228.306 minutes. For comparison, we calculate the expected travel time without considering the re-routing option and find the result as 232.514 minutes. Thus, we obtain nearly a 1.84% better solution over the no re-route solution. Also, if we assume that d_{ij}^B's are between $(6.2{*}d_{ij}^U)$ and $(6.6{*}d_{ij}^U)$, the reduction in the expected travel times becomes 7%. While calculating the expected travel times, we also find the paths for two cases. From these two results, we conclude that the paths also differ from each other. Figure 4.1 and Figure 4.2 illustrate the resulted paths. Also, if we draw the two paths on the same graph, the difference can easily be seen as in Figure 4.3. In this figure, the blue lines represent the path found in the case of no re-routing whereas the red lines represent the path found by applying our algorithm.

Figure 4.1: Illustration of The Path Considering The Re-routing Option

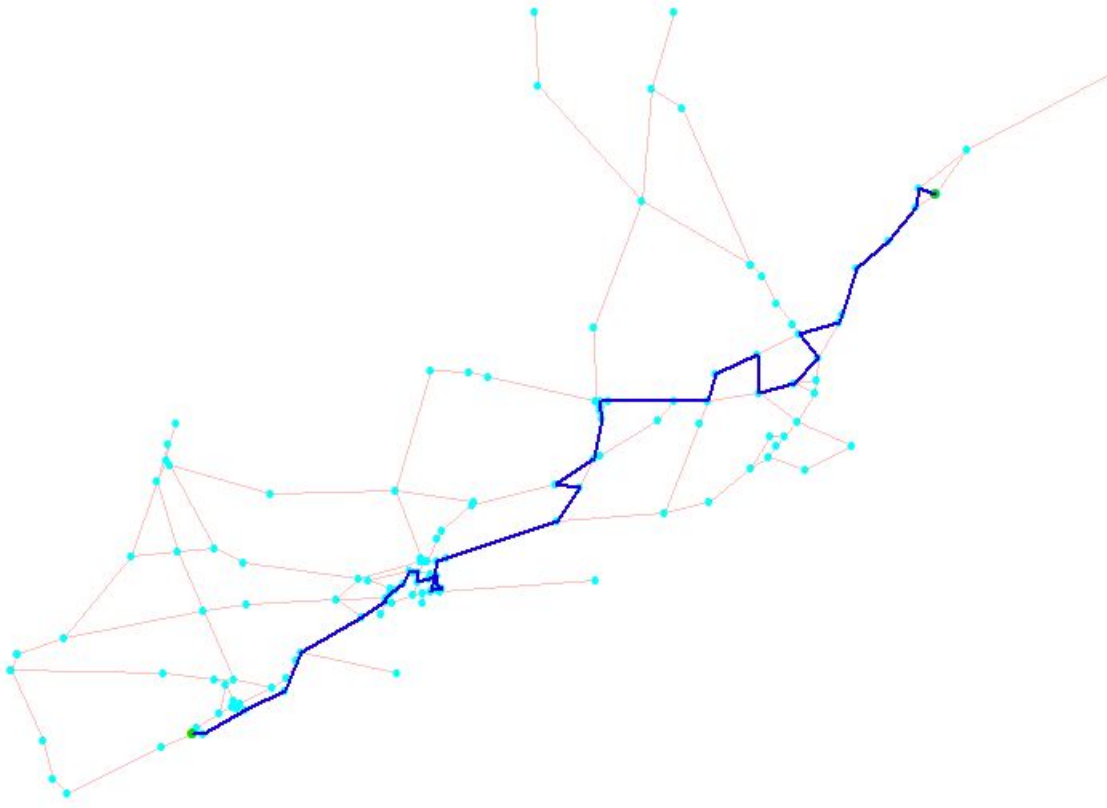

Figure 4.2: Illustration of The Path Considering No Re-routing

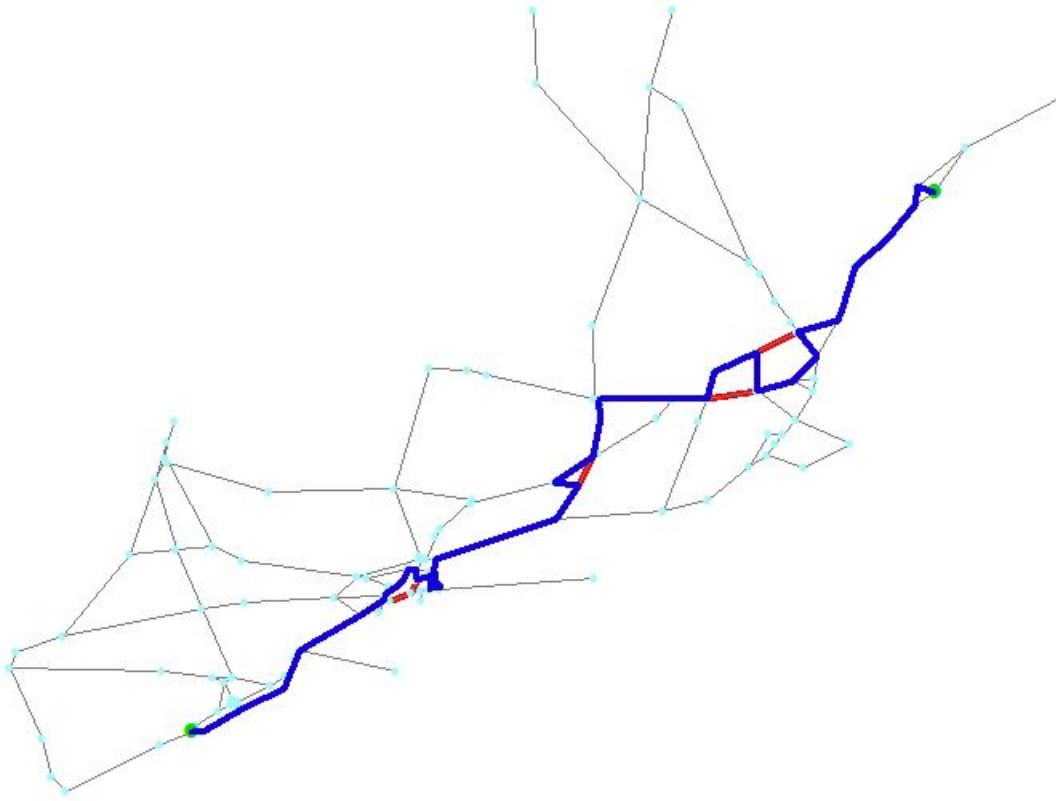

Figure 4.3: Illustration of The Resulted Paths on the same graph

From our analysis, we obtain the information about our traversal along the given network. For instance, we say that if we have an incident occurred along the arc (11, 138), we choose to re-route from that arc afterwards instead of waiting along that arc. Finally, the computational performance of our algorithm is nearly 35 seconds on a computer of 2.2 GHz speed and 2048 MByte memory. However, when we assume that there may occur at most $k = 2$ incidents, we see that the running time of the algorithm is increased to nearly 35 minutes.

As a result, we conclude that our algorithm considering the re-routing option produces better solutions and give different paths to follow when we compare with the case of no re-routing. However, we see that the proposed algorithm is not polymomial but exponential in the number of incidents and arcs as stated in Chapter 3.

4.2 Numerical Studies

In this section, we present a quantitative analysis on some different versions of our problem. We explain some of analytical results that we obtain in order to understand the effect of our algorithm under different types of the networks.

We use a test network which has 49 nodes and 130 arcs as a base network. All parameters of the network are given in Appendix B.1. We then randomly generate different problem instances based on this specific network for our numerical analysis. These instances are constructed by changing the parameters of probability of the arc (i, j) being blocked while traversing it, p_{ij}, and travel time of the arc (i, j) provided that it is blocked, d_{ij}^B. Based on these problem instances,

we aim to obtain useful insights about our algorithm and how the proposed algorithm is related to these problem parameters in general. For all our randomly generated instances, we use 20 observations to be analyzed. Also, we assume that there may occur at most $k = 1$ incident along the network.

Firstly, we randomly generate 20 observations from the base network which has p_{ij}'s differring between [0-0.50] and d_{ij}^B's differring between [(3.2*d_{ij}^U) - 3.6*d_{ij}^U)]. We number these observations as R1, R2, ..., R20. The probability of road blockage, p_{ij}, that we define as PRB, and the travel time of the arc (i, j) provided that it is blocked, d_{ij}^B, defined as DB can be considered as the two indicators of the network "robustness". Having smaller incident probabilities or shorter travel times to clear a blocked road yield more robust networks, which is equivalent to obtaining paths with less expected travel times (or any other performance metric). We analyze the impact of parameters PRB and DB on the network robustness. For this, we modify the p_{ij} values of each arc in each problem instance by a given margin and keep all other problem parameters same in order to obtain another set of problem instances with a different network robustness. Each problem instance modified in this way is denoted by PRBX_N when each p_{ij} value of the Nth original problem are reduced by specified X percentage. We randomly generate 20 observations for each modified (perturbed) problems.
Similarly, we can also obtain different problem instances by perturbing the d_{ij}^B values of the original network. Let DBX denote the set of problem instances obtained by perturbing the d_{ij}^B values of the original problem instances by X % and by keeping all other problem parameters constant.
We analyze the results obtained from these problem instances to understand the effect of the specified parameters. To compare the results, we calculate the percent change (in %) of difference between the expected travel times of our algorithm

and the algorithm considering no re-route option, i.e. regular algorithm. These are defined as follows:

ETT_SPReroute = expected travel time obtained by our algorithm
ETT_SPnoreroute = expected travel time obtained by regular algorithm

$$\%Change = ((ETT\_SPReroute) - (ETT\_SPnoreroute))/((ETT\_SPnoreroute)) * 100 \tag{4.1}$$

Hence, this % Change corresponds to the improvements obtained by (i) re-routing the vehicles en-route if and when necessary, and (ii) choosing the optimal path for the re-routing option. The results for the case of randomly generated 20 observations from the base network having p_{ij}'s differring between [0-0.50] and d_{ij}^B's differring between [(3.2*d_{ij}^U) - 3.6*d_{ij}^U)] are given in the following table:

Table 4.1: The Results of Networks with $p_{ij} \in [0, 0.5]$ and $d_{ij}^B \in [3.2d_{ij}^U, 3.6d_{ij}^U]$

Replication ♯	Standard Result	Algorithm's Result	Percent Change(%)
1	4.69	4.67	- 0.52
2	3.65	3.65	0.00
3	4.16	4.16	0.00
4	4.04	4.04	0.00
5	4.03	4.01	- 0.26
6	4.22	4.21	- 0.25
7	4.15	4.14	- 0.29
8	4.03	4.02	- 0.09
9	4.72	4.68	- 0.77
10	4.01	3.99	- 0.49
11	4.42	4.41	- 0.25
12	4.53	4.51	- 0.37
13	4.35	4.31	- 0.96
14	4.84	4.81	- 0.64
15	4.42	4.40	- 0.46
16	4.63	4.61	- 0.46
17	4.03	4.01	- 0.68
18	4.12	4.10	- 0.33
19	4.13	4.12	- 0.21
20	4.33	4.32	- 0.09
AVERAGE			**-0.36**

In Table 4.1, "Standard Result" is same as defined by ETT_SPnoreroute and "Algorithm's Result" is defined as ETT_SPReroute. From Table 4.1, we can observe that there is a slight reduction in the expected travel times of the observations when we apply our algorithm. On the average, this reduction is nearly 0.36% from all $n = 20$ samples. While calculating the expected travel times, we also find the resulted paths for these cases, and observe that no different paths are obtained in the samples as performed in the case of analyzing the real example problem in the previous section.

As stated above, to understand the effects of our problem parameters p_{ij} and d_{ij}^B seperately, we randomly construct $n = 20$ observations by changing one parameter while keeping the other fixed and observe the results obtained from these samples. For instance, we generate two different problem instances of the above example while keeping the parameter d_{ij}^B fixed. For the first version, we construct the instances as reducing p_{ij}'s by 30% and the second problem instance is constructed by reducing p_{ij}'s 60%. Similarly, we randomly generate three different instances by changing d_{ij}^B while keeping the parameter p_{ij} fixed. Finally, we compare the results to understand the effects of our algorithm.

The Effects of p_{ij}

To see the effects of the probability of the arc (i, j) being blocked while traversing it in the problem, we compare the results of different versions having different probabilities. As stated above, we randomly generate two different versions of the network problem which has p_{ij}'s between [0-0.5] and d_{ij}^B's between [(3.2*d_{ij}^U)-3.6*d_{ij}^U)] while keeping d_{ij}^B's fixed. Thus, first version is generated by reducing p_{ij}'s 30%, so the probabilities differ between [0-0.35]. The second version of the problem is generated by reducing p_{ij}'s 60%, so the probabilities differ between

[0-0.20]. The results are given in Tables 4.2 and 4.3.

Table 4.2: The Results of Networks with $p_{ij} \in [0-0.35]$ and $d_{ij}^B \in [d_{ij}^U * (3.2-3.6)]$

Replication ♯	Standard Result	Algorithm's Result	Percent Change(%)
1	3.53	3.52	- 0.25
2	3.61	3.60	- 0.04
3	4.37	4.35	- 0.39
4	4.19	4.18	- 0.12
5	4.04	4.04	0.00
6	4.45	4.39	- 1.36
7	4.02	4.02	0.00
8	4.21	4.18	- 0.66
9	4.14	4.10	- 0.76
10	3.92	3.92	0.00
11	4.05	4.04	- 0.22
12	3.54	3.53	- 0.04
13	3.50	3.49	- 0.42
14	4.11	4.11	- 0.02
15	4.05	4.04	- 0.34
16	3.85	3.84	- 0.33
17	4.23	4.22	- 0.05
18	3.92	3.91	- 0.43
19	3.63	3.62	- 0.38
20	4.54	4.53	- 0.23
AVERAGE			**-0.30**

Table 4.3: The Results of Networks with $p_{ij} \in [0-0.20]$ and $d_{ij}^{B} \in [d_{ij}^{U} * (3.2-3.6)]$

Replication ♯	Standard Result	Algorithm's Result	Percent Change(%)
1	3.84	3.82	- 0.51
2	3.28	3.27	- 0.33
3	3.74	3.73	- 0.06
4	3.62	3.60	- 0.43
5	3.53	3.53	- 0.03
6	3.82	3.79	- 0.62
7	3.87	3.86	- 0.28
8	3.31	3.31	0.00
9	3.41	3.41	- 0.00
10	3.43	3.43	0.00
11	3.80	3.76	- 0.96
12	3.81	3.79	- 0.52
13	3.81	3.78	- 0.78
14	3.89	3.88	- 0.09
15	3.77	3.77	0.00
16	3.57	3.57	- 0.04
17	3.43	3.43	- 0.16
18	3.29	3.29	- 0.06
19	3.59	3.57	- 0.31
20	3.45	3.45	- 0.08
AVERAGE			**-0.26**

From Tables 4.2 and 4.3, we can observe that the expected travel times differ from each other. Also, the difference in the percent changes for the case with $p_{ij} \in [0 - 0.35]$ is a bit higher than the one with $p_{ij} \in [0 - 0.20]$. On the average, the total reduction in the expected travel time for the first problem instance is nearly 0.30% and for the second problem instance it is nearly 0.26%. We find this reduction as nearly 0.36% for the base case having $p_{ij} \in [0 - 0.50]$. As a result, when these two instances are compared to the case with $[0 - 0.50]$, we can observe that the total reduction decreases as the probability of blocking decreases. It is found same as we expect since the re-routing option may not be chosen if the probability of blocking is nearly zero. While calculating the expected travel times, we also find the resulted paths for these problem instances, and we see that no different paths are obtained for these three cases.

The Effects of d_{ij}^B

To see the effects of the travel time of the arc (i, j) provided that it is blocked in the problem, we compare the results of different problem intances having different travel times. As in the above analysis, we randomly generate three different intances of the network problem that has p_{ij}'s between [0-0.5] and d_{ij}^B's between [(3.2*d_{ij}^U) - 3.6*d_{ij}^U)] while keeping p_{ij}'s fixed. For the first instance, we randomly generate $n = 20$ samples by reducing d_{ij}^B's 30%, thus, the travel times differ between [(2.24*d_{ij}^U) - 2.52*d_{ij}^U)]. The second problem instance is generated by increasing d_{ij}^B's 30%, so the travel times differ between [(4.16*d_{ij}^U) - 4.68*d_{ij}^U)]. The last version of the problem is generated by increasing d_{ij}^B's 60%, so the travel times differ between [(5.12*d_{ij}^U) - 5.76*d_{ij}^U)]. For the first instance, it is observed that there is no difference between the expected travel times for "Standard Result" and "Algorithm's Result". Thus, we can conclude that if the travel times of the arc (i, j) provided that it is blocked decrease, we do not obtain a better solution

in general. For more detailed analysis, the results of second and third problem instances are given in Tables 4.4 and 4.5. Also, optimal paths obtained from those problem instances by applying the algorithms considering both re-routing and no re-routing options are given in Tables 4.6 and 4.7.

Table 4.4: The Results of Networks with $p_{ij} \in [0-0.5]$ and $d_{ij}^B \in [d_{ij}^U * (4.16-4.68)]$

Replication ♯	Standard Result	Algorithm's Result	Percent Change(%)
1	5.63	5.00	- 11.23
2	4.19	4.15	- 1.06
3	4.95	4.67	- 5.76
4	4.02	3.88	- 3.38
5	4.77	4.55	- 4.79
6	4.96	4.22	- 14.93
7	4.50	4.28	- 4.77
8	4.54	4.37	- 3.69
9	4.23	4.18	- 1.28
10	4.41	3.82	- 13.44
11	5.48	4.13	- 24.62
12	5.44	4.92	- 9.67
13	4.17	4.07	- 2.21
14	4.68	3.76	-19.73
15	5.20	3.92	- 24.59
16	4.09	3.81	- 6.92
17	5.09	4.19	- 17.63
18	4.10	4.06	- 0.90
19	5.79	4.94	-14.64
20	4.47	4.15	-7.19
AVERAGE			**-9.62**

Table 4.5: The Results of Networks with $p_{ij} \in [0 - 0.50]$ and $d_{ij}^B \in [d_{ij}^U * (5.12 - 5.76)]$

Replication ♯	Standard Result	Algorithm's Result	Percent Change(%)
1	4.75	4.31	-9.21
2	5.14	4.55	-11.43
3	6.11	4.08	-33.18
4	4.69	4.37	-6.85
5	5.89	4.97	-15.59
6	6.26	4.11	-34.35
7	6.24	5.18	-17.12
8	3.80	3.60	-5.38
9	6.19	3.87	-37.42
10	4.11	3.99	-2.94
11	5.53	4.72	-14.57
12	5.77	4.91	-14.92
13	5.02	4.50	-10.39
14	4.32	4.02	-6.90
15	4.09	3.96	-3.22
16	4.90	4.54	-7.51
17	4.69	4.37	-6.82
18	6.32	4.12	-34.84
19	6.10	3.98	-34.67
20	6.19	4.56	-26.31
AVERAGE			**-16.68**

Table 4.6: The Results of Paths obtained for networks with $p_{ij} \in [0 - 0.5]$ and $d_{ij}^{B} \in [d_{ij}^{U} * (4.16 - 4.68)]$

Replication ♯	Standard Path	Algorithm's Path	Different(X)
1	0-8-25-24-48	0-2-32-30-24-48	X
2	0-8-25-24-48	0-8-25-24-48	
3	0-8-25-24-48	0-8-25-24-48	
4	0-8-25-24-48	0-8-25-24-48	
5	0-8-25-24-48	0-8-25-24-48	
6	0-8-25-24-48	0-8-25-24-48	
7	0-8-25-24-48	0-1-39-41-38-11-9-8-25-24-48	X
8	0-8-25-24-48	0-8-25-24-48	
9	0-8-25-24-48	0-8-25-24-48	
10	0-8-25-24-48	0-8-25-24-48	
11	0-1-4-2-32-30-24-48	0-1-4-2-32-30-24-48	
12	0-8-25-24-48	0-1-39-41-38-37-28-46-31-30-24-48	X
13	0-8-25-24-48	0-8-25-24-48	
14	0-8-25-24-48	0-8-25-24-48	
15	0-1-39-41-38-13-35-48	0-1-39-41-38-13-35-48	
16	0-8-25-24-48	0-1-4-2-32-30-24-48	X
17	0-8-25-24-48	0-8-25-24-48	
18	0-8-25-24-48	0-1-39-41-38-13-35-48	X
19	0-8-25-24-48	0-8-25-24-48	
20	0-8-25-24-48	0-8-9-11-38-13-35-48	X

Table 4.7: The Results of Paths obtained for networks with $p_{ij} \in [0 - 0.50]$ and $d_{ij}^{B} \in [d_{ij}^{U} * (5.12 - 5.76)]$

Replication ♯	Standard Path	Algorithm's Path	Different(X)
1	0-8-25-24-48	0-8-25-24-48	
2	0-8-25-24-48	0-8-25-24-48	
3	0-8-25-24-48	0-1-39-41-38-11-9-8-25-24-48	X
4	0-8-25-24-48	0-8-25-24-48	
5	0-8-25-24-48	0-8-25-24-48	
6	0-8-25-24-48	0-1-39-41-38-13-35-48	X
7	0-8-25-24-48	0-8-25-24-48	
8	0-8-25-24-48	0-8-25-24-48	
9	0-8-25-24-48	0-1-39-41-38-11-31-30-24-48	X
10	0-8-25-24-48	0-8-25-24-48	
11	0-8-25-24-48	0-8-25-24-48	
12	0-8-25-24-48	0-8-25-24-48	
13	0-8-25-24-48	0-8-25-24-48	
14	0-8-25-24-48	0-8-25-24-48	
15	0-8-25-24-48	0-8-25-24-48	
16	0-8-25-24-48	0-8-25-24-48	
17	0-8-25-24-48	0-8-25-24-48	
18	0-8-25-24-48	0-1-4-2-32-30-24-48	X
19	0-8-25-24-48	0-1-39-41-38-13-35-48	X
20	0-8-25-24-48	0-1-4-2-32-30-24-48	X

From Tables 4.4 and 4.5, we can observe that the difference in the expected travel times for the problem instance with $d_{ij}^B \in (d_{ij}^U * (5.12 - 5.76))$ is much higher than the one with $d_{ij}^B \in (d_{ij}^U * (4.16 - 4.68))$. From Table 4.4, we can observe that the total reduction in the expected travel times is nearly 16.68% on the average. Also, we conclude that the total reduction in the expected travel times is nearly 9.62% from the Table 4.5. For the base network having $d_{ij}^B \in (d_{ij}^U * (3.2 - 3.6))$ and $p_{ij} \in [0 - 0.50]$, we find this reduction as nearly 0.36%. As a result, when we compare these three problem instances, we observe that the total reduction increases as the travel times of the arc (i, j) provided that it is blocked increase for our problem. That is to say, our algorithm which considers the re-routing option produces much better results. Also, another important result are gathered from this analysis. From Tables 4.6 and 4.7, we can observe that when this specified travel times are increased, different paths are obtained for the sample networks in our problem. For problem instances having $d_{ij}^B \in (d_{ij}^U * (4.16 - 4.68))$ and $d_{ij}^B \in (d_{ij}^U * (5.12 - 5.76))$ with $p_{ij} \in [0 - 0.50]$, 6 replications have different optimal paths as a result. However, as stated above, we observe that no different paths are obtained for problem instance where d_{ij}^B's between [(3.2*d_{ij}^U)-3.6*d_{ij}^U)] with $p_{ij} \in [0 - 0.50]$. .

After calculating the percent changes for each 20 observations, we observe some statistics such as mean value, maximum and minimum values of the differences etc. The following table represents the results. In Tables 4.8 and 4.9, PRBX_AVE and DBX_AVE denote the average differences between the expected travel times of our algorithm and the algorithm considering no re-route option of the original 20 test problems and that of modified (perturbed) problems. This table summarizes the impact of PRB and DB on network robustness. The detail calculations in this table is given in AppendixB.2.

Table 4.8: Summary Statistics

	Average Improvement	Max.	Min.	Variance
PRB0_AVE	-0.36	-0.96	0.00	0.072
PRB-30_AVE	-0.30	-1.36	0.00	0.111
PRB-60_AVE	-0.26	-0.96	0.00	0.084
	Average Improvement	Max.	Min.	Variance
DB-30_AVE	0.00	0.00	0.00	0.00
DB0_AVE	-0.36	-0.96	0.00	0.07
DB30_AVE	-9.62	-24.62	-0.90	58.39
DB60_AVE	-16.68	-37.42	-0.29	145

Table 4.9: The Effects of Parameters p_{ij} and d_{ij}^B

INTERACTION	DB-30_AVE	DB0_AVE	DB30_AVE	DB60_AVE
PRB0_AVE	0.00	0.36	9.62	16.68
PRB-30_AVE	0.00	0.30	5.61	13.95
PRB-60_AVE	0.00	0.26	3.15	9.25

From Table 4.8, we observe that the most improvement in the expected travel time is obtained when we increase d_{ij}^B's 60%, i.e., the travel times differ between [(5.12*d_{ij}^U)- 5.76*d_{ij}^U)] while the probability p_{ij} keep fixed. Also, for this problem instance, we may obtain nearly 37.5% improvement in the expected travel time. When we analyze the effects of the related parameters on our problem simultaneously, we obtain very important results. For instance, from Table 4.9 we realize that the most improvement in the expected travel time is obtained when we increase d_{ij}^B's 60% whereas p_{ij}'s are decreased by 30% at the same time. That is, the travel times of the arc (i, j) provided that it is blocked differ between [(5.12*d_{ij}^U)-5.76*d_{ij}^U)] and the probability of the arc (i, j) being blocked while traversing it differs betwen between [0-0.35]. The improvement for this problem instance is found as 13.95%. Also, from Table 4.9, we observe that the least increase is obtained when p_{ij}'s are decreased by 60% and d_{ij}^B's are increased 30%.

As a result, we conclude that our problem parameters p_{ij} and d_{ij}^B have an

important effect in our problem. When the probability of the arc (i, j) being blocked while traversing it decreases, the expected travel times are decreased and we say that we obtain slightly better solutions in general. However, if the travel time of the arc (i, j) provided that it is blocked increases, we observe much better solutions when we apply our algorithm. For this problem instance, our algorithm gives different paths as well as in the real example problem. Also, we conclude that the interaction effect between the parameters p_{ij} and d_{ij}^B is significant from our quantitative analysis.

Chapter 5

Conclusion and Future Research

In this study, we have examined the shortest path problem under the possibility of re-routing when an arc that is being traversed is blocked due to the reasons such as road conditions, weather conditions, congestion, release of a hazmat etc. We aim to implement the possible re-routing decisions en-route, in the selection process of the path in terms of a performance measure. Particularly we have focused on the problem of finding the shortest path that minimizes the expected travel time under the possibility of re-routing. Then, we evaluate the options as the vehicle either waits until all effects of the incident are cleared and then follows the same path thereafter, or returns to the starting node of that arc and follows an escape route to the destination node, i.e. re-routing option. Also, we assume that when an incident occurs on the arc being traversed along the network (i.e. when the arc is blocked), this arc will not be visited again if re-routing alternative is chosen. That is, the arc is removed from the network when the accident occurs and the alternative of not following this arc after the event is chosen. We propose a labeling algorithm to solve for this specific problem and

model the proposed algorithm. A real case example is solved by applying the algorithm and quantitative analysis has been performed based on the real case. Several numerical studies are conducted in order to demonstrate the effectiveness of the algorithm and to assess the sensivity of the problem parameters.

We observe that evaluating of the re-routing alternative in transportation networks can lead to substantial savings. Also, we believe that our study may have important applications for logistics environments. In addition, a research may be performed to design and implement the re-routed shortest path algorithms solved in polynomial time as a future study.

Bibliography

[1] G. Andreatta and L. Romeo. Stochastic shortest paths with recourse. *Networks*, 18:193–204, 1988.

[2] V. J. Blue, J. L. Adler, and G. F. List. Real-time multiple-objective path search for in-vehicle route guidance systems. *Transportation Research Record: Journal of the Transportation Research Board*, 1588, 1997.

[3] T. Chang, L. K. Nozick, and M. A. Turnquist. Multiobjective path finding in stochastic dynamic networks, with application to routing hazardous materials shipments. *Transportation Science*, 39(3):383–399, 2005.

[4] R. K. Cheung. Iterative methods for dynamic stochastic shortest path problems. *Naval Research Logistics*, 45(8):769–789, 1998.

[5] E. W. Dijkstra. A note on two problems on connexion with graphs. *Numer.Math.*, 1:395–412, 1959.

[6] E. Erkut and O. Alp. Integrated routing and scheduling of hazmat trucks with stops en route. *Transportation Science*, 41(1):107–122, 2007.

[7] E. Erkut and S. A. Tjandra. Multiobjective routing in stochastic and time-varying networks with real-time updates. November 2004.

[8] L. Fu. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transportation Research Part B*, 35:749–765, 2001.

[9] L. Fu and L. R. Rilett. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research-B*, 32(7):499–516, 1998.

[10] S. Gao and I. Chabini. Optimal routing policy problems in stochastic time-dependent networks. *Transportation Research Part B*, 40:93–122, 2006.

[11] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151:1–11, 2003.

[12] R. W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[13] G. F. List, P. B. Mirchandani, M. A. Turnquist, and K. G. Zografos. Modeling and analysis for hazardous materials transportation:risk analysis, routing/scheduling and facility location. *Transportation Science*, 25(2):100–114, 1991.

[14] E. Miler-Hooks. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks*, 37(1):35–52, 2001.

[15] E. Miller-Hooks and H. Mahmassani. Path comparisons for a priori and time-adaptive decisions in stochastic, time-varying networks. *European Journal of Operational Research*, 146:67–82, 2003.

[16] E. Miller-Hooks and H. S. Mahmassani. Optimal routing of hazardous materials in stochastic, time-varying transportation networks. *Transportation Research Record*, 1645(98-0416):143–151, 1998b.

[17] L. R. Nielsen, K. A. Andersen, and D. Pretolani. Bicriterion a priori route choice in stochastic time-dependent networks. Working Paper L-2006-10. Accessed at http://www.hha.dk/bs/wp/log/L-2006-10.pdf, 2006.

[18] S. Opasanon and E. Miller-Hooks. Adjustable preference path strategies for use in multicriteria, stochastic, and time-varying transportation networks. *Transportation Research Record: Journal of the Transportation Research Board*, 1923 / 2005:137–143, 2005.

[19] G. H. Polychronopoulos and J. N. Tsitsiklis. Stochastic shortest path problems with recourse. *Networks*, 27:133–143, 1996.

[20] J. S. Provan. A polynomial-time algorithm to find shortest paths with recourse. *Networks*, 41(2):115–125, 2003.

[21] H. N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61:143–164, 1995.

[22] S. T. Waller and A. K. Ziliaskopoulos. On the online shortest path problem with limited arc cost dependencies. *Networks*, 40(4):216–227, 2002.

[23] A. B. Wijeratne, M. A. Turnquist, and P. B. Mirchandani. Multiobjective routing of hazardous materials in stochastic networks. *European Journal of Operational Research*, 65(1):33–43, 1993.

Appendix A

ALGORITHMS PART

A.1 The Algorithm of Shortest Path Problem with re-routing en-route

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define NOTVISITED 0
#define VISITED 1
typedef struct ConnectionDefinition {
        struct ConnectionDefinition *left;
        struct ConnectionDefinition *right;
        int node;
        float costwithoutcrash;                                    10
        float costwithcrash;
        float crashprobability;
} ConDefinition;
typedef struct NodeDefinition {
        ConDefinition *connections;
        int nodeindex;
        int state;
        float distance;
        int previous;
        int queueindex;                                            20
```

```
} NodeDefinition;
typedef struct ConnectionList {
        int connectioncount;
        int *list;
} ConnectionList;
char* shortestfilename= NULL;
char* historyfilename=NULL;
char* graphfolder=NULL;
char written = 'f';
char uselookup = 'f';                                                    30
void addAsLeaf(ConDefinition *cparent, ConDefinition *condef);
void addNodeToBST(NodeDefinition *nodegraph, ConDefinition *condef,
int sourcenode);
ConDefinition *recurseLookup(ConDefinition *cparent, int dnode);
ConDefinition
*lookupConnection(NodeDefinition *nodegraph, int snode, int dnode);
void dumpBST(ConDefinition *cparent);
void dumpGraph(NodeDefinition *nodegraph, int nodecount);
void dumpSettings(int crashes, int nodecount, int sourcenode,
int destinationnode);                                                    40
void dumpConnections(ConnectionList *edgelists, int nodecount);
void dumpAllPairsShortest(float **distances, int nodecount);
float min(float x, float y);
void allPairsShortest(float **distances, NodeDefinition *nodegraph,
ConnectionList *edgelist, int nodecount);
void dumpPriorityQueue(NodeDefinition **pqueue, int queueLength);
void dumpShortestsWithExcludes(float **shortestwithexcludes, int nodecount,
int destinationnode);
void heapify(NodeDefinition **array, int length);
void addNodeToPriorityQueue(NodeDefinition **array, NodeDefinition *newNode, 50
int *queueLength);
void changeDistance(NodeDefinition **minQueue, NodeDefinition *nodegraph,
int nodeindex, float value);
NodeDefinition *getMinimum(NodeDefinition **array, int *queueLength);
void copyBST(NodeDefinition *nodegraph, int source, ConDefinition *root,
int excsource, int excdest);
NodeDefinition *cloneAndReduceGraph(NodeDefinition *ngraph, int excsource,
int excdestination, int nodecount);
ConnectionList *cloneAndReduceEdgeList(ConnectionList *elists, int excsource,
int excdest, int nodecount);                                             60
float expectedPathCost(int crashes, int nodecount, int ***previous, int j,
int i, int destinationnode, NodeDefinition *nodegraph,
ConnectionList *edgelists, int excsource, int excdest, float ***, char);
float expectedTravel(int ***previous, NodeDefinition **pqueue,
NodeDefinition *nodegraph, ConnectionList *edgelists, int nodecount,
int source, int destination, int crashes, int *queueLength,
```

```
int excsource, int excdest, float ***, char);
float calculateSubGraph(NodeDefinition *nodegraph, ConnectionList *edgelists,
int crashes, int sourcenode, int destinationnode, int nodecount,
int excsource, int excdest, char isStandard);
void freeConnectionList(ConnectionList *con);
void freeBST(ConDefinition *root);
void freeNodeGraph(NodeDefinition *ngraph, int nodecount);
void freeLookup(float ***lookup, int nodecount);
void freePrevious(int ***previous, int nodecount);
void freeQueue(NodeDefinition **pqueue, int queueLength);
void freeDistances(float **distances, int nodecount);
void addAsLeaf(ConDefinition *cparent, ConDefinition *condef) {
if (cparent->node < condef->node) {
if (cparent->right == NULL)
cparent->right = condef;
else
addAsLeaf(cparent->right, condef);
} else {
if (cparent->left == NULL)
cparent->left = condef;
else
addAsLeaf(cparent->left, condef);
}
return;
}
void addNodeToBST(NodeDefinition *nodegraph, ConDefinition *condef,
int sourcenode) {
if (nodegraph[sourcenode].connections == NULL) {
nodegraph[sourcenode].connections = condef;
} else {
ConDefinition *cparent = nodegraph[sourcenode].connections;
addAsLeaf(cparent, condef);
}
return;
}
ConDefinition *recurseLookup(ConDefinition *cparent, int dnode) {
if (cparent->node == dnode) {
return cparent;
} else if (cparent->node < dnode) {
if (cparent->right == NULL)
return NULL;
else
return recurseLookup(cparent->right, dnode);
} else {
if (cparent->left == NULL)
return NULL;
```

```
else
return recurseLookup(cparent->left, dnode);
}
}
ConDefinition *lookupConnection(NodeDefinition *nodegraph, int snode, int dnode) {
if (nodegraph[snode].connections == NULL) {
return NULL;
} else {
ConDefinition *cparent = nodegraph[snode].connections;
return recurseLookup(cparent, dnode);
}
}
void dumpBST(ConDefinition *cparent) {
if (cparent->left != NULL) {
dumpBST(cparent->left);
}
printf("%d [%f %f %f]\n", cparent->node, cparent->costwithcrash,
cparent->costwithoutcrash, cparent->crashprobability);
if (cparent->right != NULL) {
dumpBST(cparent->right);
}
return;
}
void dumpGraph(NodeDefinition *nodegraph, int nodecount) {
int i = 0;
printf("\nConnections (Binary Search Tree)\n");
for (i=0; i<nodecount; i++) {
printf("Source Node : %d ---> ", i);
printf("Destinations : ");
if (nodegraph[i].connections == NULL) {
printf("None\n");
continue;
} else {
dumpBST(nodegraph[i].connections);
}
printf("\n");
}
}
void dumpSettings(int crashes, int nodecount, int sourcenode,
int destinationnode) {
printf("Max. Crash Count : %d\n", crashes);
printf("Node Count : %d\n", nodecount);
printf("Source : %d\n", sourcenode);
printf("Destination : %d\n\n", destinationnode);
}
void dumpConnections(ConnectionList *edgelists, int nodecount) {
```

```
int i = 0;
int j = 0;
printf("\nConnections (Listed)\n");
for (i=0; i<nodecount; i++) {
if (edgelists[i].connectioncount == 0) {
printf("Source : %d Destinations : None\n", i);
} else {
int *lst = edgelists[i].list;
printf("Source : %d Destinations : ", i);
int count = edgelists[i].connectioncount;
for (j=0; j<count; j++) {
printf("%d ", lst[j]);
}
printf("\n");
}
}
}
void dumpAllPairsShortest(float **distances, int nodecount) {
int i, j;
float current;
printf("\nAll-Pairs Shortest Paths : \n");
for (i=0; i<nodecount; i++) {
for (j=0; j<nodecount; j++) {
printf("D(%d,%d) = ", i, j);
current = distances[i][j];
if (current == INT_MAX) {
printf("Inf\n");
} else {
printf("%f\n", current);
}
}
}
}
void dumpAllPairsShortestToFile(float **distances, int nodecount, char* filename) {
int i, j;
float current;
FILE *file;
file = fopen(filename, "w");
fprintf(file, "\nAll-Pairs Shortest Paths : \n");
for (i=0; i<nodecount; i++) {
for (j=0; j<nodecount; j++) {
fprintf(file, "D(%d,%d) = ", i, j);
current = distances[i][j];
if (current == INT_MAX) {
fprintf(file, "Inf\n");
} else {
```

```
fprintf(file, "%f\n", current);
}
}
}
fclose(file);
}
float min(float x, float y) {
return (x<y) ? x : y;
}
void freeDistances(float **distances, int nodecount) {
int i;
for (i = 0; i < nodecount; ++i) {
float *ptr = distances[i];
free(ptr);
}
free(distances);
}
void allPairsShortest(float **distances, NodeDefinition *nodegraph,
ConnectionList *edgelist, int nodecount) {
int i = 0, count=0, j=0, k=0, destination, source;
for (i=0; i<nodecount; i++) {
distances[i][i] = 0.0f;
}
for (i=0; i<nodecount; i++) {
if (edgelist[i].connectioncount == 0)
continue;
count = edgelist[i].connectioncount;
for (j=0; j<count; j++) {
source = i;
destination = (edgelist[i].list)[j];
ConDefinition *spec = lookupConnection(nodegraph, source,
destination);
distances[source][destination] = spec->costwithoutcrash;
}
}
for (k=0; k<nodecount; k++) {
for (i=0; i<nodecount; i++) {
for (j=0; j<nodecount; j++) {
distances[i][j] = min(distances[i][j], distances[i][k]
+ distances[k][j]);
}
}
}
return;
}
void standartExpectation(float **distances, NodeDefinition *nodegraph,
```

```
ConnectionList *edgelist, int nodecount) {
int i = 0, count=0, j=0, k=0, destination, source;
for (i=0; i<nodecount; i++) {
distances[i][i] = 0.0f;
}
for (i=0; i<nodecount; i++) {
if (edgelist[i].connectioncount == 0)
continue;
count = edgelist[i].connectioncount;
for (j=0; j<count; j++) {                                    260
source = i;
destination = (edgelist[i].list)[j];
ConDefinition *spec = lookupConnection(nodegraph, source,
destination);
distances[source][destination] = ((1−spec−>crashprobability)
* spec−>costwithoutcrash) + (spec−>crashprobability
* spec−>costwithcrash);
}
}
for (k=0; k<nodecount; k++) {                                270
for (i=0; i<nodecount; i++) {
for (j=0; j<nodecount; j++) {
distances[i][j] = min(distances[i][j], distances[i][k]
+ distances[k][j]);
}
}
}
return;
}
void freeQueue(NodeDefinition **pqueue, int queueLength) {   280
int i =0;
for (i = 0; i<queueLength; i++) {
free(pqueue[i]);
}
free(pqueue);
}
void dumpPriorityQueue(NodeDefinition **pqueue, int queueLength) {
int i =0;
for (i = 0; i<queueLength; i++) {
if (pqueue[i] == NULL)                                       290
printf("%d --> NULL\n", i);
else
printf("%d --> %d\n", i, pqueue[i]−>queueindex);
}
}
void dumpShortestsWithExcludes(float **shortestwithexcludes, int nodecount,
```

```
int destinationnode) {
int i, j = 0;
for (i=0; i<nodecount; i++) {
for (j=0; j<nodecount; j++) {
if (i != j) {
if (shortestwithexcludes[i][j]!= INT_MAX) {
printf("Shortest %d -> %d ex: %d -> %d = %f\n", i,
destinationnode, i, j, shortestwithexcludes[i][j]);
} else {
printf("Shortest %d -> %d ex: %d -> %d = INF.\n", i,
destinationnode, i, j);
}
}
}
}
}
void heapify(NodeDefinition **array, int length) {
int pindex;
int current = length - 1;
while (current > 0) {
pindex = (current - 1) / 2;
NodeDefinition *child = array[current];
NodeDefinition *parent = array[pindex];
if (child->distance > parent->distance)
break;
child->queueindex = pindex;
parent->queueindex = current;
NodeDefinition *tmp = child;
array[current] = array[pindex];
array[pindex] = tmp;
tmp=NULL;
current = pindex;
}
}
void addNodeToPriorityQueue(NodeDefinition **array, NodeDefinition *newNode,
int *queueLength) {
if (*queueLength == 0) {
newNode->queueindex = 0;
array[0] = newNode;
*queueLength = (*queueLength) + 1;
} else {
newNode->queueindex = *queueLength;
array[*queueLength] = newNode;
*queueLength = (*queueLength) + 1;
heapify(array, *queueLength);
}
```

```
}
void changeDistance(NodeDefinition **minQueue, NodeDefinition *nodegraph,
int nodeindex, float value) {
nodegraph[nodeindex].distance = value;
heapify(minQueue, (nodegraph[nodeindex].queueindex) + 1);
}
NodeDefinition *getMinimum(NodeDefinition **array, int *queueLength) {
if (*queueLength == 0)                                                    350
return NULL;
NodeDefinition *first = array[0];
array[0] = array[*queueLength - 1];
array[0]->queueindex = 0;
array[*queueLength - 1] = NULL;
*queueLength = (*queueLength) - 1;
int leftchild;
int rightchild;
int current = 0;
int min;                                                                  360
while (current < *queueLength) {
rightchild = (current + 1) * 2;
leftchild = rightchild - 1;
NodeDefinition *left= NULL;
NodeDefinition *right= NULL;
if (leftchild < *queueLength) {
left = array[leftchild];
}
if (rightchild < *queueLength) {
right = array[rightchild];                                                370
}
if (left == NULL && right == NULL)
break;
if (right == NULL)
min = leftchild;
else if (left->distance <= right->distance)
min = leftchild;
else
min = rightchild;
NodeDefinition *minchild = array[min];                                    380
if (minchild->distance >= array[current]->distance)
break;
minchild->queueindex = current;
array[current]->queueindex = min;
NodeDefinition *tmp = minchild;
array[min] = array[current];
array[current] = tmp;
tmp=NULL;
```

```
current = min;
}
return first;
}
void dijkstra(NodeDefinition **pqueue, NodeDefinition *nodegraph,
ConnectionList *edgelists, int nodecount, int source, int destination,
int excludeneighbour, int *queueLength) {
int i=0;
float ndist = 0;
for (i=0; i<nodecount; i++) {
if (i != source)
nodegraph[i].distance = INT_MAX;
else
nodegraph[i].distance = 0;
nodegraph[i].previous = -1;
nodegraph[i].queueindex = -1;
nodegraph[i].state = NOTVISITED;
addNodeToPriorityQueue(pqueue, &(nodegraph[i]), queueLength);
}
//dumpPriorityQueue (pqueue, *queueLength);
while (*queueLength > 0) {
NodeDefinition *u = getMinimum(pqueue, queueLength);
if (u->nodeindex == destination)
break;
ConnectionList conlist = edgelists[u->nodeindex];
int *neighbours = conlist.list;
for (i=0; i<conlist.connectioncount; i++) {
NodeDefinition *current = &nodegraph[neighbours[i]];
if (current->state != VISITED) {
if (u->nodeindex == source && current->nodeindex
== excludeneighbour)
continue;
ConDefinition *spec = lookupConnection(nodegraph, u->nodeindex,
neighbours[i]);
ndist = u->distance + spec->costwithoutcrash;
if (ndist < current->distance) {
changeDistance(pqueue, nodegraph, current->nodeindex, ndist);
current->previous = u->nodeindex;
}
}
}
u->state = VISITED;
}
for (i=0; i<nodecount; i++) {
nodegraph[i].state = NOTVISITED;
}
```

```
}
void freePrevious(int ***previous, int nodecount) {
int i, j;
for (i = 0; i < nodecount; ++i) {
for (j = 0; j < nodecount; ++j) {
free(previous[i][j]);
}
free(previous[i]);
}
free(previous);
}
void dijkstraWithExcludes(int ***previous, float **shortestwithexcludes,
NodeDefinition **minQueue, NodeDefinition *nodegraph,
ConnectionList *edgelists, int nodecount, int destinationnode,
int *queueLength) {
int i, j, mid;
for (i=0; i<nodecount; i++) {
for (j=0; j<nodecount; j++) {
if (i == j)
continue;
ConDefinition *spec = lookupConnection(nodegraph, i, j);
if (spec != NULL) {
*queueLength = 0;
dijkstra(minQueue, nodegraph, edgelists, nodecount, i,
destinationnode, j, queueLength);
shortestwithexcludes[i][j]
= nodegraph[destinationnode].distance;
mid = destinationnode;
for (; i < mid;) {
previous[i][j][mid] = nodegraph[mid].previous;
mid = nodegraph[mid].previous;
}
}
}
}
}
void copyBST(NodeDefinition *nodegraph, int source, ConDefinition *root,
int excsource, int excdest) {
if (root != NULL) {
if ((excsource<0 && excdest<0) || ((source != excsource || root->node
!=excdest) && (source != excdest || root->node!=excsource) )) {
ConDefinition *condef;
condef = (ConDefinition *) malloc(sizeof(ConDefinition));
condef->left = NULL;
condef->right = NULL;
condef->node = root->node;
```

```
condef->costwithoutcrash = root->costwithoutcrash;
condef->costwithcrash = root->costwithcrash;
condef->crashprobability = root->crashprobability;
addNodeToBST(nodegraph, condef, source);
}
copyBST(nodegraph, source, root->left, excsource, excdest);
copyBST(nodegraph, source, root->right, excsource, excdest);
}
}
NodeDefinition *cloneAndReduceGraph(NodeDefinition *ngraph, int excsource,
int excdestination, int nodecount) {
int i;
NodeDefinition *nodegraph =
(NodeDefinition *) malloc(sizeof(NodeDefinition) * nodecount);
for (i=0; i<nodecount; i++) {
NodeDefinition ndef;
ndef.connections = NULL;
ndef.nodeindex = i;
ndef.state = NOTVISITED;
ndef.distance = INT_MAX;
ndef.previous = -1;
ndef.queueindex = -1;
nodegraph[i] = ndef;
//if ((excsource<0 && excdestination<0) || (i != excsource && i != excdestination)) {
ConDefinition *root = ngraph[i].connections;
if (root != NULL) {
copyBST(nodegraph, i, root, excsource, excdestination);
}
//}
}
return nodegraph;
}
ConnectionList *cloneAndReduceEdgeList(ConnectionList *elists, int excsource,
int excdest, int nodecount) {
ConnectionList *edgelists;
int connectioncount = 0;
int i, j, count;
edgelists = (ConnectionList *) malloc(sizeof(ConnectionList) * nodecount);
for (i=0; i<nodecount; i++) {
edgelists[i].connectioncount = 0;
edgelists[i].list = NULL;
connectioncount = 0;
//if ((excsource<0 && excdest<0) || (i != excsource || i!= excdest)) {
for (j=0; j<elists[i].connectioncount; j++) {
if ((excsource<0 || excdest<0) || ((i!=excsource
&& i!=excdest) || (elists[i].list[j]!=excdest
```

```
&& elists[i].list[j]!=excsource))) {
connectioncount++;
}
}
if (connectioncount > 0) {
edgelists[i].connectioncount = connectioncount;
edgelists[i].list = (int *) malloc(sizeof(int) * (connectioncount));
count = 0;
for (j=0; j<elists[i].connectioncount; j++) {
if ((excsource<0 && excdest<0) || ((i!=excsource
&& i!=excdest) || (elists[i].list[j]!=excdest
&& elists[i].list[j]!=excsource))) {
if (i != elists[i].list[j]) {
edgelists[i].list[count] = elists[i].list[j];
count++;
}
}
}
}
//}
}
return edgelists;
}
void freeLookup(float ***lookup, int nodecount) {
int i, j;
for (i = 0; i < nodecount; ++i) {
for (j = 0; j < nodecount; ++j) {
free(lookup[i][j]);
}
free(lookup[i]);
}
free(lookup);
}
float expectedPathCost(int crashes, int nodecount, int ***previous, int j,
int i, int destinationnode, NodeDefinition *nodegraph,
ConnectionList *edgelists, int excsource, int excdest,
float ***lookup_iZk, char isStandard) {
float pij, dij, jZk_1, dijB, iZ_ij, jZk, iZk, min1=0, min2=0, min=0;
int m, n;
char reroute;
FILE *file;
FILE *graphviz;
ConDefinition *spec = lookupConnection(nodegraph, i, j);
NodeDefinition *nodegraph_clone;
ConnectionList *edgelists_clone;
char graphvizfilename [100];
```

```
char script[200];
char *commonFileName;
if (spec == NULL) {
printf("Spec is NULL!i : %d j : %d \n", i, j);
return 0;
} else {
}
pij = (spec->crashprobability);
dij = (spec->costwithoutcrash);
        /* Eger onceden hesaplanmissa tekrar hesaplama*/
if (uselookup == 't' && lookup_iZk[i][j][crashes] != 0) {
printf("Using lookup value for iZk, i:%d , j:%d , k:%d\n", i, j,
crashes);
//return lookup_iZk[i][j][crashes];
}
if (crashes == 0) {
if (j == destinationnode) {
jZk = 0;
} else if (nodegraph[j].distance != INT_MAX) {
jZk = nodegraph[j].distance;
} else {
printf("There is a problem here 1!\n");
//jZk = expectedTravel(degree, j, i, destinationnode, nodegraph);
}
iZk = dij + jZk;
} else {
dijB = spec->costwithcrash;
if (j == destinationnode) {
jZk_1 = 0;
if (isStandard != 's') {
printf("%d\n\n", j);
printf(
"1.Clonning NodeGraph excluding %d - %d for degree %d. . .\n",
i, j, crashes - 1);
nodegraph_clone = cloneAndReduceGraph(nodegraph, i, j,
nodecount);
printf(
"1.Clonning EdgeList excluding %d - %d for degree %d. . .\n",
i, j, crashes - 1);
edgelists_clone = cloneAndReduceEdgeList(edgelists, i, j,
nodecount);
printf(
"1.Calculating (|%d - %d|/|%d - %d|)(k-1) for degree %d. . .\n",
i, destinationnode, i, j, crashes - 1);
iZ_ij = calculateSubGraph(nodegraph_clone, edgelists_clone,
crashes -1, i, destinationnode, nodecount, i, j,
```

```
isStandard);
}
} else {
nodegraph_clone = cloneAndReduceGraph(nodegraph, −1, −1, nodecount);
edgelists_clone = cloneAndReduceEdgeList(edgelists, −1, −1,
nodecount);
printf("2.Calculating (|%d - %d|)(k-1) for degree %d. . .\n", j,
destinationnode, crashes − 1);
jZk_1 = calculateSubGraph(nodegraph_clone, edgelists_clone, crashes−1, j,
destinationnode, nodecount, −1, −1, isStandard);
if (isStandard != 's') {
printf
"2.Clonning NodeGraph excluding %d - %d for degree %d. . .\n",
i, j, crashes − 1);
nodegraph_clone = cloneAndReduceGraph(nodegraph, i, j,
nodecount);
printf(
"2.Clonning EdgeList excluding %d - %d for degree %d. . .\n",
i, j, crashes − 1);
edgelists_clone = cloneAndReduceEdgeList(edgelists, i, j,
nodecount);
printf(
"2.Calculating (|%d - %d|/|%d - %d|)(k-1) for degree %d. . .\n",
i, destinationnode, i, j, crashes − 1);
iZ_ij = calculateSubGraph(nodegraph_clone, edgelists_clone,
crashes −1, i, destinationnode, nodecount, i, j,
isStandard);
}
}
if (j == destinationnode) {
jZk = 0;
} else if (nodegraph[j].distance != INT_MAX) {
jZk = nodegraph[j].distance;
} else {
printf("There is a problem here 1!\n");
//jZk = expectedTravel(degree, j, i, destinationnode, nodegraph);
}
min1 = dijB + jZk_1;
if (isStandard != 's') {
min2 = dij + iZ_ij;
} else {
min2 = INT_MAX;
}
if (min1 < min2)
min = min1;
else
```

```
min = min2;
iZk = (1 - pij)*(dij + jZk) + pij*min;
}
if (nodegraph[i].distance > iZk) {
if ((historyfilename != NULL && isStandard != 's') || (shortestfilename
!= NULL && isStandard == 's')) {
if (isStandard != 's') {
commonFileName = historyfilename;
} else {
commonFileName = shortestfilename;
}
file = fopen(commonFileName, "a");
if (min1 <= min2) {
reroute = 's';
} else {
reroute = 'r';
}
if (written == 'f') {
written = 't';
fprintf(file, "%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n",
"crashes", "source", "destination", "iZk", "min1",
"min2", "reroute", "excs", "excd", "path");
}
if (graphfolder != NULL) {
sprintf(graphvizfilename, "%s/%d_%d_%d_%f_%f_%f_%c_%d_%d.txt",
graphfolder, crashes, i, j, iZk, min1, min2, reroute,
excsource, excdest);
fprintf(file, "%d\t%d\t%d\t%f\t%f\t%f\t%c\t%d\t%d\t%s\n",
crashes, i, j, iZk, min1, min2, reroute, excsource,
excdest, graphvizfilename);
graphviz = fopen(graphvizfilename, "w");
fprintf(graphviz, "graph G {\n");
for (m=0; m<nodecount; m++) {
if (edgelists[m].connectioncount == 0) {
continue;
} else {
int *lst = edgelists[m].list;
int count = edgelists[m].connectioncount;
for (n=0; n<count; n++) {
if (lst[n]>m) {
if (((m == i && lst[n] == j) || (m == j
&& lst[n] == i))) {
fprintf(graphviz,
"%d -- %d [style = bold];\n", m,
lst[n]);
} else {
```

```
fprintf(graphviz, "%d -- %d;\n", m, lst[n]);
}
}
}
}
}
fprintf(graphviz, "}");
fclose(graphviz);
sprintf(script, "neato -Tjpg %s -o %s.jpg", graphvizfilename,
graphvizfilename);
        system(script);
} else {
fprintf(file, "%d\t%d\t%d\t%f\t%f\t%f\t%c\t%d\t%d\t%s\n",
crashes, i, j, iZk, min1, min2, reroute, excsource,
excdest, "NA");
}
fclose(file);
}
}
if (uselookup == 't') {
lookup_iZk[i][j][crashes] = iZk;
}
return iZk;
}
float expectedTravel(int ***previous, NodeDefinition **pqueue,
NodeDefinition *nodegraph, ConnectionList *edgelists, int nodecount,
int source, int destination, int crashes, int *queueLength,
int excsource, int excdest, float ***lookup_iZk, char isStandard) {
int i=0, j=0;
float ndist = 0;
float result = 0;
for (i=0; i<nodecount; i++) {
if (i != destination)
nodegraph[i].distance = INT_MAX;
else
nodegraph[i].distance = 0;
nodegraph[i].previous = -1;
nodegraph[i].queueindex = -1;
nodegraph[i].state = NOTVISITED;
addNodeToPriorityQueue(pqueue, &(nodegraph[i]), queueLength);
}
while (*queueLength > 0) {
NodeDefinition *u = getMinimum(pqueue, queueLength);
int *neighbours = edgelists[u->nodeindex].list;
int ncount = edgelists[u->nodeindex].connectioncount;
for (j=0; j<ncount; j++) {
```

```
NodeDefinition neighbour = nodegraph[neighbours[j]];
if (neighbour.state != VISITED) {
//printf("NOT VISITED : %d\n", neighbour.nodeindex);
ndist = expectedPathCost(crashes, nodecount, previous,                    760
u->nodeindex, neighbour.nodeindex, destination,
nodegraph, edgelists, excsource, excdest, lookup_iZk,
isStandard);
ndist, neighbour.distance);
if (ndist < neighbour.distance) {
printf("\nMinimum Distance [%d %d] Updated to %f\n\n",
neighbour.nodeindex, destination, ndist);
changeDistance(pqueue, nodegraph, neighbour.nodeindex,
ndist);
neighbour.previous = u->nodeindex;                                        770
}
} else {
;
}
}
u->state = VISITED;
}
result = nodegraph[source].distance;
freeNodeGraph(nodegraph, nodecount);
freeConnectionList(edgelists);                                            780
freePrevious(previous, nodecount);
if (uselookup == 't') {
freeLookup(lookup_iZk, nodecount);
}
freeQueue(pqueue, *queueLength);
free(queueLength);
//      for (i=0; i<nodecount; i++) {
//              nodegraph[i].state = NOTVISITED;
//      }
        return result;                                                    790
}
float calculateSubGraph(NodeDefinition *nodegraph, ConnectionList *edgelists,
int crashes, int sourcenode, int destinationnode, int nodecount,
int excsource, int excdest, char isStandard) {
NodeDefinition **minQueue;
int i, j, k;
int *queueLength = (int *) malloc(sizeof(int));
*queueLength = 0;
int ***previous;
float exptravel = INT_MAX;                                                800
float ***lookup_iZk;
previous = (int ***) malloc(sizeof(int **) * nodecount);
```

```
for (i=0; i<nodecount; i++) {
previous[i] = (int **) malloc(sizeof(int *) * nodecount);
for (j=0; j<nodecount; j++) {
previous[i][j] = (int *) malloc(sizeof(int) * nodecount);
for (k=0; k<nodecount; k++) {
previous[i][j][k] = i;
}
}
}
minQueue = (NodeDefinition **) malloc(sizeof(NodeDefinition *) * nodecount);
for (i=0; i<nodecount; i++) {
minQueue[i] = (NodeDefinition *) malloc(sizeof(NodeDefinition));
minQueue[i] = NULL;
}
*queueLength = 0;
if (uselookup == 't') {
lookup_iZk = (float ***) malloc(sizeof(float **) * nodecount);
for (i=0; i<nodecount; i++) {
lookup_iZk[i] = (float **) malloc(sizeof(float *) * nodecount);
for (j=0; j<nodecount; j++) {
lookup_iZk[i][j]
= (float *) malloc(sizeof(float) * (crashes+1));
for (k=0; k<crashes+1; k++) {
lookup_iZk[i][j][k] = 0;
}
}
}
}
*queueLength = 0;
exptravel = expectedTravel(previous, minQueue, nodegraph, edgelists,
nodecount, sourcenode, destinationnode, crashes, queueLength,
excsource, excdest, lookup_iZk, isStandard);
printf("Expected Cost %d --> %d = %f\n", sourcenode, destinationnode,
exptravel);
return exptravel;
}
void freeNodeGraph(NodeDefinition *ngraph, int nodecount) {
int i;
for (i = 0; i < nodecount; i++) {
ConDefinition *root = ngraph[i].connections;
freeBST(root);
}
free(ngraph);
}
void freeBST(ConDefinition *root) {
if (root != NULL) {
```

```
freeBST(root->left);
freeBST(root->right);
free(root);
}
}
void freeConnectionList(ConnectionList *con) {
free(con->list);
free(con);
}
int main(int args, char **argv) {
if (args != 6 && args != 5 && args != 4 && args != 3 && args != 2) {
printf("Usage : ./a.out <filename> [<standartresultsfile:null>] "
"[<decisiveresultsfile:null>] [<graphsfolder:null>] [<uselookuptable:null>]");
return -1;
}
FILE * networkFile;
char c;
int snode;
int dnode;
float costcrash;
float costnocrash;
float prob;
int scan;
int i;
int prevsource = -1;
int connectioncount;
int crashes, nodecount, sourcenode, destinationnode;
NodeDefinition *nodegraph_clone;
ConnectionList *edgelists_clone;
NodeDefinition *nodegraph;
ConnectionList *edgelists;
float finalResult;
networkFile=fopen(argv[1], "r");
if (networkFile==NULL) {
perror("Error opening file");
return -1;
} else {
fscanf(networkFile, "%c %d ", &c, &crashes);
fscanf(networkFile, "%c %d ", &c, &nodecount);
fscanf(networkFile, "%c %d ", &c, &sourcenode);
fscanf(networkFile, "%c %d ", &c, &destinationnode);
nodegraph = (NodeDefinition *) malloc(sizeof(NodeDefinition)
* nodecount);
for (i=0; i<nodecount; i++) {
NodeDefinition ndef;
ndef.connections = NULL;
```

```
ndef.nodeindex = i;
ndef.state = NOTVISITED;
ndef.distance = INT_MAX;
ndef.previous = −1;
ndef.queueindex = −1;
nodegraph[i] = ndef;                                                    900
}
edgelists = (ConnectionList *) malloc(sizeof(ConnectionList)
* nodecount);
for (i=0; i<nodecount; i++) {
fscanf(networkFile, "%d %d ", &snode, &connectioncount);
if (connectioncount > 0) {
edgelists[snode].connectioncount = connectioncount;
edgelists[snode].list = (int *) malloc(sizeof(int)
* (connectioncount));
} else {                                                                910
edgelists[snode].connectioncount = 0;
edgelists[snode].list = NULL;
}
}
printf("\n");
i=0;
while ((scan=fscanf(networkFile, "%d %d %f %f %f ", &snode, &dnode,
&costnocrash, &costcrash, &prob))!=EOF) {
if (prevsource != snode) {
i = 0;                                                                  920
} else {
i++;
}
costcrash, prob);
ConDefinition *condef;
condef = (ConDefinition *) malloc(sizeof(ConDefinition));
condef−>left = NULL;
condef−>right = NULL;
condef−>node = dnode;
condef−>costwithoutcrash = costnocrash;                                 930
condef−>costwithcrash = costcrash;
condef−>crashprobability = prob;
printf("WithCrash = %f , WithoutCrash = %f , Probability = %f\n",
costcrash, costnocrash, prob);
addNodeToBST(nodegraph, condef, snode);
(edgelists[snode].list)[i] = dnode;
prevsource = snode;
};
fclose(networkFile);
printf("Argument Count : %d\n", args);                                  940
```

```
if (args > 2) {
if (strcmp(argv[2], "null") != 0) {
printf("Calculating Standard Expectation to : %s", argv[2]);
shortestfilename = argv[2];
}
if (args > 3) {
if (strcmp(argv[3], "null") != 0) {
printf("Calculating Decisive Expectation to : %s", argv[3]);
historyfilename = argv[3];
}
if (args > 4) {
if (strcmp(argv[4], "null") != 0) {
printf("Drawing graphs to folder : %s", argv[4]);
graphfolder = argv[4];
}
if (args > 5) {
if (strcmp(argv[5], "null") != 0) {
printf("Using iZk lookup table. . .");
uselookup = 't';
}
}
}
}
}
if (shortestfilename != NULL) {
nodegraph_clone = cloneAndReduceGraph(nodegraph, -1, -1, nodecount);
edgelists_clone = cloneAndReduceEdgeList(edgelists, -1, -1,
nodecount);
finalResult = calculateSubGraph(nodegraph_clone, edgelists_clone,
crashes, sourcenode, destinationnode, nodecount, -1, -1,
's');
printf("Standard Expectation : %f\n", finalResult);
}
written = 'f';
finalResult = calculateSubGraph(nodegraph, edgelists, crashes,
sourcenode, destinationnode, nodecount, -1, -1, 'd');
printf("Decisive Expectation : %f\n", finalResult);
}
return 0;
}
```

A.2 Path Finder Algorithm

```
package banu.shortest.report;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Iterator;
import java.util.TreeMap;
import java.util.TreeSet;

public class PathFinder {
        public static void main(String[] args) {
                try {
TreeMap<Integer, TreeMap<Double, TreeSet<Integer>>>
source = new TreeMap<Integer,TreeMap<Double, TreeSet<Integer>>>();
String filename = args[0];
int crashes = Integer.parseInt(args[2]);
FileReader reader = new FileReader(new File(filename));
BufferedReader breader = new BufferedReader(reader);
String str = "";
str = breader.readLine();
while(str != null) {
str = breader.readLine();
if (str == null) break;
String[] parts = str.split("\t");
System.out.println("param:" + parts[0]);
if (crashes == Integer.parseInt(parts[0])) {
TreeMap<Double, TreeSet<Integer>> dist = source.get
(Integer.valueOf(parts[1]));
if (dist == null) {
dist = new TreeMap <Double, TreeSet<Integer>>();
}
TreeSet<Integer> dest = dist.get(Double.valueOf(parts[3]));
if (dest == null) {
dest = new TreeSet<Integer>();
}
dest.add(Integer.valueOf(parts[2]));
dist.put(Double.valueOf(parts[3]), dest);
source.put(Integer.valueOf(parts[1]), dist);
}
}
```

```
breader.close();
FileWriter fwriter = new FileWriter(new File(args[1]));
Iterator<Integer> it = source.keySet().iterator();
TreeSet<Integer> visited = new TreeSet<Integer>();
Integer first = null;
Integer prev = null;
if (it.hasNext()) {                                                    50
Integer s = it.next();
while(s.intValue() != Integer.parseInt(args[3])) {
System.out.println("Source Node: " + s);
visited.add(s);
TreeMap<Double, TreeSet<Integer>> dists = source.get(s);
Iterator<TreeSet<Integer>> itdests = dists.values().iterator();
boolean found = false;
while(itdests.hasNext()) {
TreeSet<Integer> dist = itdests.next();
Iterator<Integer> itf = dist.iterator();                               60
while (itf.hasNext()) {
first = (Integer) itf.next();
if(visited.contains(first)) {
continue;
} else {
System.out.println("Dest Node: " + first);
found = true;
break;
}
}                                                                      70
if (found) break;
}
fwriter.write(s + " " + first + " 1\n");
s = first;
}
}
//fwriter.write(args[3] + " 1");
fwriter.flush();
fwriter.close();
} catch (FileNotFoundException e) {                                    80
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}
```

Appendix B

NUMERICAL STUDIES OF CHAPTER 4

B.1 Parameters of Test Network

Table B.1: Parameters of Test Network

From	To	d_{ij}^U	d_{ij}^B	p_{ij}	From	To	d_{ij}^U	d_{ij}^B	p_{ij}
0	42	1.93	6.47	0.27	24	26	0.32	1.12	0.48
0	8	0.72	2.33	0.5	24	30	2.16	7.33	0.11
0	4	1.21	4.25	0.45	25	47	12.25	43.35	0.41
0	1	0.08	0.24	0.46	25	48	6.22	21.69	0.09
0	3	10.72	38.56	0.22	26	25	7.57	24.34	0.07
0	2	0.86	2.86	0.05	26	48	2.82	9.75	0.11
1	39	0.4	1.29	0.1	26	47	1.2	4.27	0.13
1	20	2.87	10.17	0.21	27	29	13.39	47.64	0.18
1	2	1.28	4.53	0.21	27	33	1.03	3.32	0.37
1	4	0.2	0.65	0.11	27	43	0.69	2.21	0.26
1	3	12.1	40.42	0.07	27	32	11.43	41.14	0.33
2	4	0.09	0.32	0.28	27	31	1.26	4.55	0.15
2	32	0.2	0.65	0.17	27	30	1.41	5	0.43
2	3	2.14	7.44	0.23	28	30	14.13	46.08	0.31
2	16	2.43	7.81	0.2	28	46	0.2	0.69	0.2
3	14	2.43	8.72	0.15	28	37	0.36	1.26	0.02
3	27	2.5	8.3	0.08	28	31	1.15	3.69	0.05
3	4	2.1	7.25	0.23	28	32	2.22	7.56	0.31
5	7	1.79	6.07	0.48	28	33	1.37	4.69	0.25
5	20	13.05	42.83	0.19	28	27	0.53	1.71	0.19
5	6	1.08	3.61	0.44	29	31	2.48	8.49	0.28
5	10	9.16	32.85	0.3	29	43	2.4	8.15	0.15
6	7	1.51	4.9	0.16	29	32	14.27	50.32	0.1
6	29	0.16	0.54	0.23	29	28	2.61	8.45	0.29
6	37	1.87	6.43	0.13	29	30	1.28	4.1	0.09
8	12	2.84	9.91	0.09	29	33	5.81	19.13	0.05
8	25	1.52	5.35	0.42	30	32	1.73	5.68	0.41
8	11	2.27	7.5	0.25	30	44	1.78	5.92	0.13
8	10	6.94	23.4	0.15	30	33	7.73	25.34	0.02
8	9	0.63	2.18	0.18	31	30	1.22	4.2	0.1
9	11	0.62	2	0.12	31	32	2.41	7.73	0.45

From	To	d_{ij}^U	d_{ij}^B	p_{ij}	From	To	d_{ij}^U	d_{ij}^B	p_{ij}
9	12	11.95	41.49	0.49	31	13	14.44	46.74	0.22
10	11	1.93	6.46	0.12	31	11	0.71	2.34	0.41
10	9	2.86	10.02	0.19	31	33	0.59	1.92	0.3
10	28	2.02	7.09	0.13	31	46	0.2	0.64	0.37
10	12	0.73	2.54	0.2	32	33	3.7	11.9	0.32
11	38	0.16	0.54	0.09	32	44	1.25	4.18	0.28
11	12	1.08	3.69	0.2	34	36	0.31	0.98	0.32
13	38	0.78	2.75	0.46	35	36	0.64	2.26	0.4
13	15	0.23	0.75	0.22	35	34	2.55	8.38	0.41
14	15	1.38	4.83	0.19	35	9	2.91	10.19	0.24
14	13	1.02	3.53	0.22	35	48	1.23	3.99	0.45
14	9	0.84	2.96	0.43	35	13	1	3.41	0.45
14	47	14.26	48.4	0.02	37	38	0.57	1.89	0.18
16	18	1.49	4.93	0.02	37	41	1.48	4.88	0.36
17	34	0.45	1.57	0.06	37	34	1.07	3.7	0.41
17	18	1.36	4.52	0.21	37	46	1.89	6.77	0.01
17	16	1.81	6.1	0.27	38	46	2.75	9.04	0.39
17	9	1.47	4.79	0.14	38	43	1.65	5.34	0.22
17	45	11.55	41.44	0.28	38	41	0.62	2.14	0.37
19	22	0.61	2.08	0.27	39	41	0.52	1.81	0.33
20	19	1.75	5.94	0.01	39	37	1.46	4.83	0.48
20	22	1.91	6.57	0.12	39	16	1.31	4.29	0.12
21	22	1.14	3.98	0.34	39	44	1.24	4.11	0.19
21	11	1.97	6.62	0.01	39	38	1.35	4.6	0.48
21	23	13.15	43.41	0.21	40	38	10.87	38.11	0.47
21	20	1.65	5.68	0.27	40	39	1.07	3.79	0.3
21	19	2.36	7.84	0.34	40	46	1.29	4.4	0.47
22	23	2.36	8.02	0.21	40	19	0.15	0.52	0.49
22	27	1.32	4.74	0.16	40	29	2.15	7.35	0.19
23	19	10.91	38.04	0.37	40	41	1.38	4.62	0.36
23	20	4.36	14.9	0.48	40	37	1.61	5.77	0.42
24	48	0.26	0.83	0.08	42	14	0.16	0.51	0.49
24	47	0.5	1.67	0.1	45	34	0.17	0.59	0.24
24	25	0.44	1.51	0.45	47	48	2.98	9.66	0.29

B.2 Numerical Calculations for understanding the impact of Problem Parameters

Table B.2: Numerical Calculations

PRB-30_AVE-DB30_AVE	Standard Result	Algorithm's Result	Percent Change (%)
	4.12	3.89	5.58
	4.17	4	4.08
	4.9	4.52	7.76
	4.29	4.16	3.03
	4.69	4.38	6.61
	3.7	3.66	1.08
	4.5	4.23	6.00
	4.92	4.5	8.54
	4.9	4.55	7.14
	4.75	4.45	6.32
AVERAGE			**5.61**
PRB-30_AVE-DB60_AVE	Standard Result	Algorithm's Result	Percent Change (%)
	4.8	4.23	11.87
	5.37	4.5	16.20
	5.2	4.38	15.77
	5.33	4.64	12.95
	5.32	4.22	20.68
	5.12	4.48	12.50
	5.75	4.82	16.17
	5.26	4.57	13.12
	5.12	4.23	17.38
	3.53	3.43	2.83
AVERAGE			**13.95**

PRB-60_AVE-DB30_AVE	Standard Result	Algorithm's Result	Percent Change (%)
	3.71	3.67	1.08
	3.27	3.22	1.53
	3.89	3.75	3.60
	3.83	3.74	2.35
	4.4	4.12	6.36
	3.83	3.63	5.22
	3.91	3.8	2.81
	4.34	4.12	5.07
	3.6	3.59	0.28
	3.44	3.33	3.20
AVERAGE			**3.15**
PRB-60_AVE-DB60_AVE	Standard Result	Algorithm's Result	Percent Change (%)
	4.59	4.05	11.76
	4.36	3.88	11.01
	4.43	3.9	11.96
	3.68	3.53	4.08
	4.09	3.7	9.54
	4.6	3.99	13.26
	4.23	3.81	9.93
	4.08	3.74	8.33
	3.58	3.37	5.87
	3.71	3.46	6.74
AVERAGE			**9.25**